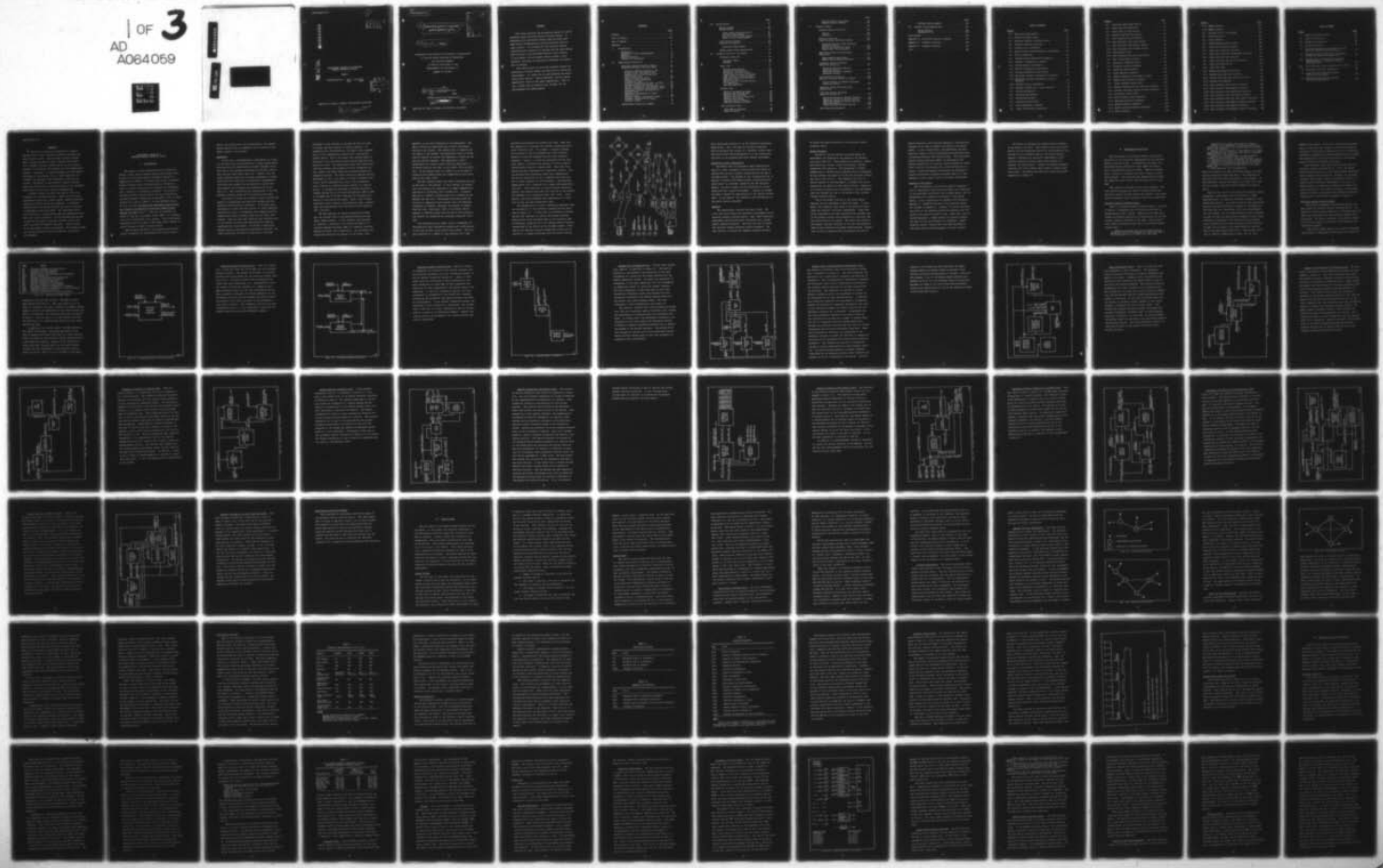
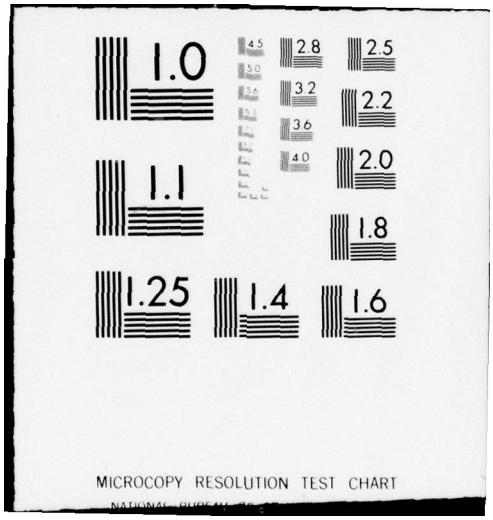
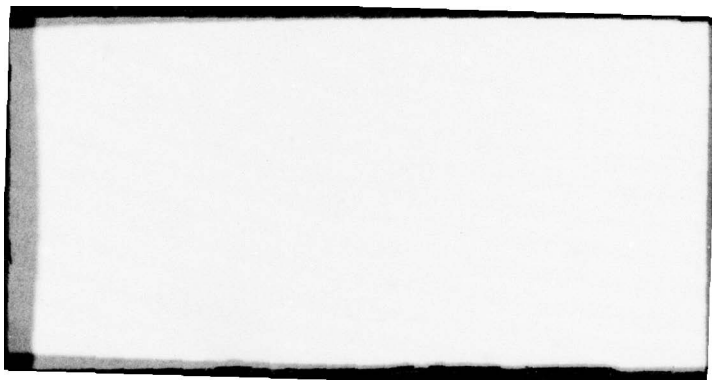
AD-A064 059 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
PRELIMINARY DESIGN OF A UNIVERSAL NETWORK INTERFACE DEVICE.(U)
DEC 78 S C SLUZEVICH

UNCLASSIFIED AFIT/GE/EE/78-41 NL

1 OF 3
AD
A064059




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART



AFIT/GE/EE/78-41

AD A064059

LEVEL II

DDC FILE COPY

PRELIMINARY DESIGN OF A UNIVERSAL NETWORK INTERFACE DEVICE

THESIS

AFIT/GE/EE/78-41 Sam C. Sluzevich
Capt USAF

Approved for public release; distribution unlimited.

19 Jan 79

JOSEPH P. HIPPS, Major, USAF
Director of Information

(14) AFIT/GE/EE/78-41

(6) # PRELIMINARY DESIGN OF A UNIVERSAL NETWORK INTERFACE DEVICE.

(9) Master's THESIS,

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air Training Command
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

(10) Sam C. Sluzevich B.S.E.E.
Capt USAF
Graduate Electrical Engineering

(11) Dec[redacted]78 (12) 274 p.

Approved for public release; distribution unlimited.

Preface

This report presents the preliminary design of a microprocessor based universal network interface device. In developing this preliminary design, many decisions were made without recommendations from the possible users of such a device. It is hoped that the universal network interface device project will continue and the users become more involved in the continued design of such a device. In this manner, the end product should provide a cost-effective interface for application throughout the Department of Defense.

This thesis would not have been possible without the assistance of several people and their help is gratefully acknowledged. Dr. Lamont was an understanding and encouraging thesis advisor. Phyllis Reynolds' skillful typing improved the looks of this paper immeasurably. Most of all, I would like to thank my wife, Belinda, for her encouragement and understanding.

Contents

	Page
Preface	ii
List of Figures	vii
List of Tables	x
Abstract	xi
I. Introduction	1
Background	2
Objective of this Investigation	2
Approach	7
Design Procedure	8
Overview of the Thesis	9
II. Requirements Definition	11
Universal Network Interface Device	11
Structured Analysis Activity Model	13
Universal Network Interface (A0)	16
Process Local Information (A1)	17
Receive Local to-be-Transmitted Information (A11)	18
Store Information (A113)	20
Process to-be-transmitted Information (A12)	21
Transmit Information to Network (A13)	22
Process Network Information (A2)	23
Receive Information from Network (A21)	24
Process Information from Network (A22)	26
Retransmit Network Information on Network (A23)	27
Retransmit Information to Local Receiver (A24)	28
Process Control Information (A243)	29
Transmit Information to Local Receiver (A244)	30
Requirements Definition Summary	31

Page

III. System Design 32

Design Dilemma 32
System Bounds 34

Local Signal Characteristics 35
I/O Port Requirements 37
Network I/O Port Requirements 38
Local I/O Port Requirements 40

Link Control Protocol 44
Function Allocation 46

Processor Requirements 51

System Design Phase Observations 54

IV. Hardware Selection and Design 55

Processor Selection 55

Processor Board 59
Z80-MCB 60

Input Card 61

RS-232C Requirements 61
Functional Requirements 62
Peripheral I/O Port Design 63
Input Board Processor Interface 65
Z80A Processor Interrupt Modes 66
Z80A Interrupt Acknowledgement 67
Z80 Daisy Chain 68
M8214 Priority Interrupt Device 70
I/O Addressing 74
Data Bus Buffering 76

Network Card 79

Network Transmission Speed 79
Word Storage Through DMA 80
Network Card With DMA 81
Network Card Device Selection 82
Network Card Design 83
Data Bus Control Design 84
Network Interface Standard 85

Dual Processor Card 88

Z80A Memory Reference 89
Basis of Design 89

Page

Dual Processor Card Design 91
Hardware Design Summary 102

V. Software Design 104

Software Design Constraint 104

Testing 105
Protocol 106

Software Functions 106
Input Processor Operating System 107

Initialization of Input Processor Operating System 110
Device Initialization Phase 110
Operational Initialization 114

Operating System #1 Generalized Subroutine 116

Queue Addition/Deletions 116
Memory Table Addition/Deletion 123

Interrupt Service Routines Operating System #1 124

Operating System #1 Receive Interrupt Routine 127
Operating System #1 Transmit Interrupt Routine 128

Main Operating System #1 134
Network Processor Operating System 135

Initialization of Network Processor Operating System 141

Operating System #2 Generalized Subroutines 142

Interrupt Service Routines Operating System #2 142

Operating System #2 Transmit Routine . . . 143
Operating System #2 Receive Routine . . . 143
Operating System #2 Special Receive Interrupt Routine 145
Timer Interrupt Service Routine 145

Main Operating System #2 145

Page

Software Design Summary 149

VI. Results and Recommendations 154

Design Results 154
Recommendations 156

Bibliography 158

Appendix A: Structured Analysis Diagrams 161

Appendix B: Hardware Circuitry 168

Appendix C: Assembled Software 189

Vita 259

List of Figures

Figure		Page
1-1	Munti-Ring Base Network	6
2-1	SA Activity Model Index	14
2-2	Universal Network Interface Device	15
2-3	Universal Network Interface	16
2-4	Process Local Information	17
2-5	Receive Local To-Be-Transmitted Information	18
2-6	Store Information	20
2-7	Process To-Be-Transmitted Information	21
2-8	Transmit Information to Network	22
2-9	Process Network Information	23
2-10	Receive Information from Network	24
2-11	Process Information from Network	26
2-12	Retransmit Network Information on Network	27
2-13	Retransmit Network Information to Local Receiver.	28
2-14	Process Control Information	29
2-15	Transmit Information to Local Receiver	30
3-1	Centralized Network	39
3-2	Decentralized Network	39
3-3	Distributed Network	41
3-4	Processor Service Time	53
4-1	Local Subscriber Interface	64
4-2	Input Card Priority Controller	71

Figure		Page
4-3	Interrupt Acknowledge Control	73
4-4	Input Card I/O Addressing	75
4-5	Input Card Data Bus Control	77
4-6	Network Card Data Bus Control	86
4-7	Dual Processor State Analysis	92
4-8	Dual Processor Card Input Design	93
4-9	Dual Processor Card Request Design	95
4-10	Dual Processor Card Refresh Control	96
4-11	Z80A Refresh Cycle	96
4-12	Dual Processor Card Data Bus Design	99
5-1	The Initialization Process	111
5-2	Subroutine ITUART Flowchart	113
5-3	Processor Lockout Mechanism	119
5-4	Operating System Lockout	120
5-5	Queue Addition Flowchart	121
5-6	Remove Information from Head of Queue	122
5-7	Memory Table Deletion Flowchart	125
5-8	Packet Sequence Control Word	128
5-9	Receive Interrupt Service Routine Flowchart	129
5-10	Transmit Interrupt Service Routine Flowchart	132
5-11	Multibuffer Status Word	134
5-12	Local Transmit Queue Flowchart	136
5-13	Operating System #1 Flowchart	139
5-14	Network Transmit Flowchart	144
5-15	Special Receive Condition Flowchart	146
5-16	Timer Flowchart	148

Figure		Page
5-17	NWTXQ Flowchart	150
5-18	NWRXQ Flowchart	151
A-1	Top-down View of an SA Model	163
A-2	Arrow Defintions	163
A-3	Arrow Branches	165
A-4	Arrows Showing Mutual Control	165
B-1	Input Card Address Circuitry	169
B-2	Input Card Interrupt Circuitry	170
B-3	Input Card Interrupt Acknowledge Circuitry	171
B-4	Input Card Data Bus Circuitry	172
B-5	Input Card Data Bus Control Circuitry	173
B-6	Input Card Z80A-CTC #2	174
B-7	Input Card 2651	175
B-8	Network Card Address Circuitry	176
B-9	Network Card Data Bus Circuitry	177
B-10	Network Card Data Bus Control Circuitry	178
B-11	Network Card Z80A-SIO	179
B-12	Dual Processor Card A12-A15 Circuitry	180
B-13	Dual Processor Card Address Circuitry	181
B-14	Dual Processor Card Read/Write Circuitry	182
B-15	Dual Processor Card Data Base Circuitry	183
B-16	Dual Processor Card Data Bus Control Circuitry.	185
B-17	Dual Processor Card Memory Request Circuitry	186
B-18	Dual Processor Card Refresh Circuitry	187
B-19	Dual Processor Card Address Control Circuitry	188

List of Tables

Table		Page
I	Protocol Characteristics	45
II	Input Card Function.	48
III	Network Card Function	48
IV	Software Function	49
V	8 Bit Microprocessors Considered for the Universal Network Interface Device	59
VI	Maximum Delay from Processor I/O Request to Data Available from Input Card	78
VII	Maximum Delay from Processor Interrupt Acknowledgement to Address Available from Input Card	78
VIII	Dual Processor Card Evaluation M1 T3/ M2 T1 .	101
IX	Processor Functional Segregation	108
X	Input Processor Operating System Functional Segregation	109
XI	Network Process Operating System Functional Segregation	140

Abstract

A preliminary design was developed for a special microprocessor based interface called the universal network interface device. The universal network interface device accepts peripheral inputs, formats these inputs into a message structure established by the link control protocol, and transmits the messages over the communication network. Conversely, it accepts messages from the network, determines if the message is for a local subscriber and transmits the message to the subscriber or back on the network. The design of the universal network interface device was modularized to allow the device to be configured based upon the user local network requirements.

A digital system life cycle was used to serve as a framework for the design project. Within the life cycle, requirements definition, system design, hardware selection/design and software design was completed. A technique patterned after Structured Analysis was used to construct a requirements definition model. The requirement model was converted to a system design model by segregating hardware and software functions. A Zilog Z80A-MCB was selected to perform the software functions and MSI circuits were used to perform the hardware functions. Circuit design of all the modular cards was developed. The software needed for the dual processor board configuration was written and assembled.

PRELIMINARY DESIGN OF A UNIVERSAL NETWORK INTERFACE DEVICE

I. Introduction

The purpose of this investigation was to design and develop a small special-purpose digital device which could be used for interfacing general peripheral devices to a communications network. The device was called a universal network interface since the device must be flexible in order to provide interfacing for a majority of peripherals into a majority of contemporary networks. The need for a universal network interface device was first proposed by the 1842 Electronic Engineering Group (EEG) of the Air Force Communication Service (AFCS) in an 1842 EEG/EEIC report, TR 78-5, entitled An Engineering Assessment Towards Economic, Feasible and Responsive Base-Level Communications Through the 1980's (Ref 1). The idea was expanded and tasked to Rome Air Development Center (RADC) for incorporation into a postdoctoral study program. This investigation represented the first phase of the study effort towards an actual universal network interface device.

The following sections of this chapter provide background information for understanding the need for such a

device, the objectives of this investigation, the general design approach that was employed, and an overview of the topics covered in this thesis.

Background

In the past, telecommunication requirements on a typical Air Force base were satisfied in a rather simple manner by providing voice communication through telephone facilities plus a few low-speed teletypewriter and data circuits over the base cable system (Ref 1:2). However, with the recent tendency toward use of digital processors to accomplish base-level functions, the base-level telecommunication facilities needed to be reevaluated to insure they could support the increased data communications needs (Ref 1:2). This reevaluation was accomplished in the 1842 EEG/EEIC technical report TR 78-5 mentioned previously.

One facet of the TR 78-5 technical report involved the method of accomplishing the base-level message and data switching and distribution functions. At the base level, distribution of data and other traffic is a most important consideration since it encompasses user terminals and the communication paths connecting them into the local area network. There are more user terminals than anything else in the network and thus costs associated with them are multiplied by a large factor. To satisfy to base-level message and data switching and distribution functions, the report postulated the need to connect any of the base

processors to any terminal on the base and also the need to connect any base terminal to another terminal. To accomplish this interconnection, the report first proposed use of a star communication network with a centralized digital switch. Each of the base's data devices would be connected to the centralized switch through dedicated communication lines. The centralized switch could then establish the necessary interconnectivity plus accomplish any code, speed, and format conversion necessary between non-compatible devices. The report noted this approach had disadvantages in that it was costly in terms of network flexibility, switch implementation, and in the transmission costs involved in connecting every terminal to a central switch. An attempt was then made to develop alternative schemes through the use of direct multiplexing (FDM or TDM) or an ALOHA (Ref 2:362-387) technique for connecting the devices to the centralized switch. Again, each of these techniques, while reducing interconnection costs, injected their own disadvantages into the central switch approach (Ref 1:162-163).

The next approach the report considered was based upon the concepts used in the Advanced Research Project Agency (ARPA) network. In this network, each processor or terminal is connected to the network by means of an Interface Message Processor (IMP) or a Terminal Interface Message Processor (TIP) respectively. The processors and stand-alone terminals can operate in any format, code,

protocol, or bit rate convenient to the subscribers. The IMP or TIP has two interfaces--one to their subscribers and one to the network. The network side is standard with all others in the network; the subscriber's side is customized as required to convert the subscriber's traffic to and from the network standard. The subscriber's side is asynchronous (it accepts traffic from the subscriber on a bit-by-bit basis at any rate, in any format, with any protocol). On the network side, all traffic is in the form of fixed bit-size packets which are transmitted at high transmission rate (Ref 1:164).

The last network concept the report considered was that of the loop or ring network. In this concept, all processors and terminals are connected to a common communication path which is configured into a closed ring. A terminal desiring to transmit a message does so by transmitting the message onto the ring. The message continues around the ring and is repeated by each terminal until the addressee recognizes its address, whereupon the message is removed from the ring. Here again, there is no central switch; however, an IMP/TIP concept must be used at each processor and terminal to accomplish any necessary conversion (Ref 1: 164).

The configuration the report finally recommended for the base-level data distribution network was a modification to the ring concept called a multi-ring network. This network consisted of a number of ring networks with a mode

providing interconnectivity between the rings. Each ring was composed of a processor and terminals associated with a given functional area. For example, a logistic terminal was connected to the ring composed by a base processor housing its data and programs and with other logistics terminals. Figure 1-1, which was extracted from the technical report, illustrates the concepts of such a group of interconnected rings on a typical base. This multi-ring concept had many advantages for a base-level network. From the Automated Data Processing (ADP) side, communication control was simplified in that a communication front end was no longer required for the processors. The processors communicated to all terminals via a simple high-speed multiplex port. Since there was no central-control switch/processors, the network costs were only incurred as the network was expanded on an incremental basis (Ref 1:165). Transmission costs were minimized since subscribers on a given ring utilize the same transmission cable.

The implementation of such a network was dependent upon the availability of the different interface devices specified in Figure 1-1. In this case, five different interface devices were necessary to realize the network selected. These devices would accomplish most of the functions such as buffering, packeting and a rate change function normally accomplished by the IMP or TIP in the ARPA network. Since each of the interface devices accomplished a basic set of functions, it seemed conceivable that one device could be

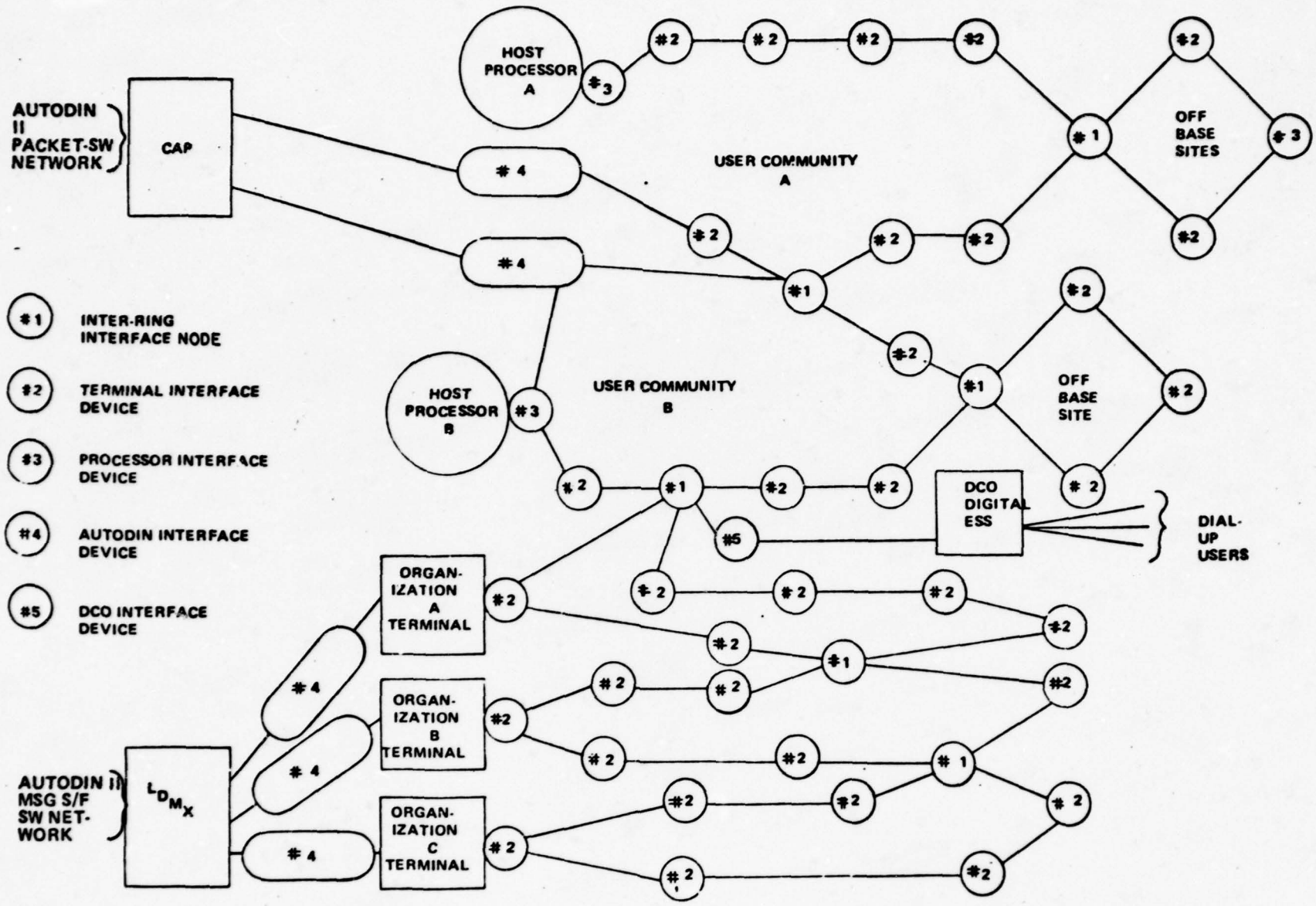


Fig. 1-1. Multi-Ring Base Network

built which would satisfy all of the different interfacing applications. Thus, the need for different interface devices generated the concept of a universal network interface device which would accomplish all of the interfacing functions in the proposed base-level network environment.

Objective of this Investigation

The thesis topic as proposed by RADC enumerated the need for a small economical interface/switching device to perform the functions normally accomplished by the IMP in the ARPA network. The purpose of this investigation was to develop a flexible microcomputer-based interfacing device which would, as a minimum, accomplish the IMP functions. However, the device design was not restricted to the proposed base-level network environment. Instead, an attempt was made to expand the applicability to any network environment. In this manner, the flexibility and universality of the device would be extended.

Approach

The investigation involved four major tasks. The first task was to define the functional requirements of the universal network interface device. Next, these functional requirement specifications were translated into a system design (hardware/software). The third task was to design the universal network interface device's hardware. The last task was to develop the computer programs necessary

to verify the proper operation of tne universal network interface device.

Design Procedure

One way to view the process of system design and development was suggested by the phases of the software life cycle (Ref 3:5). These phases are conception, requirement definition, design, coding and checkout, testing, integration and operation. The progression of phases demonstrates a top-down design approach which is considered by many software designers to be the most efficient development cycle (Ref 4:12-24). The software life cycle is usually applied to the development of software but can be generalized and applied to most design efforts. Because of its top-down structure and its generality, this development cycle was selected for use in the design of the universal network interface device.

One of the phases involved in the chosen design approach was the requirements definition phase. In this phase, the conceptual ideas about a new system are translated into specific functional requirements. These functional requirements are then verified by the ultimate user of the system to insure they accomplish all functions that were envisioned for the system. This phase thus involves the system's designer conveying to the user the designer's ideas on what functions the system should perform. Because this is such an important phase and because any form of

English expression involves some ambiguity, a more definite language can be used to support this phase of the design effort. In this investigation, the methodology and documentation chosen to define requirements were patterned after a Structured Analysis (SA) activity model. This Structured Analysis Design Technique (SADT) was developed by the SofTech Corporation as a precise, graphic method for identifying functions and showing their interrelationship in a system. Structured Analysis conventions are described in several publications produced by SofTech (Refs 5; 6) and Appendix A gives a short review of the major conventions.

Overview of the Thesis

This investigation involved the complete documentation of requirements definition using SADT, the translation of the requirement definition into a system design, and the implementation in hardware and software of the system design. Circuit designs for all hardware are provided in Appendix B. Assembled versions of the operating system are included in Appendix C. However, in certain instances, the action of the operating system was dependent upon the network link control protocol in use. Since this would be network-dependent, a dummy network link control protocol was used to allow the program to be assembled. Sample interrupt service routines have also been developed to facilitate easier user development of actual routines.

The thesis is arranged into chapters which correlate to the design life cycle. This chapter serves as an introduction with the background portion of the introduction correlating to the conceptual phase of the design process. Chapter II develops the functional requirements of the universal network interface device while Chapter III develops the system design. Chapter IV discusses hardware selection and circuit design while Chapter V details software design. The thesis concludes with results and recommendations in Chapter VI.

II. Requirements Definition

The second phase of the design process involved the definition of the functional requirements for the universal network interface device. To accomplish this phase, the Structured Analysis Design Technique (SADT) was used to build a requirements definition model. SADT was selected after a review of a previous work (Ref 7) which utilized this technique. This previous work demonstrated the modular simplicity which results from the application of the SADT.

This chapter is divided into two major sections. The first section develops the specific functions which the universal network interface device must perform. The latter section translates these functional tasks into a SADT model.

Universal Network Interface Device

What is a universal network interface device? A review of Figure 1-1 suggested certain functions which must be accomplished by such a device. Nodes #1 and #2 were envisioned as performing basically as concentrators for the subscriber terminals connected to the nodes. Martin (Ref 8: 314) listed the following functions for a hold-and-forward concentrator:

> Buffering messages from the low-speed terminal subscriber lines for transmission in modified form on the high-speed line (or lines) and vice versa.

Allocation of storage and control of queues.

Receipt and transmission of messages on the low-speed lines, using the line control procedure appropriate for the terminal.

Receipt and transmission of messages on the higher-speed network lines, using the line control procedure appropriate for the computer.

Polling the low-speed lines if they are multi-dropped or controlled by a loop configuration.

Converting the code if necessary from that used by the terminal to that used on the line to the computer.

Conversion of start-stop transmission on the low-speed line to synchronous transmission on the high-speed line.

Error detection and retransmission.

There was, however, one basic difference between the nodes and the concentrator described above. The nodes must be concerned with the routing information in the message. If this additional function was added to the above list, then the list became a good functional breakdown for nodes #1 and #2.

Nodes #3, #4, and #5 in the worst case situation would accomplish a concentrator function identical to those described above. In addition, each must accomplish a very specialized function. For nodes #3 and #5, this specialized function involved interfacing a computer or telephone system (with their own input/output (I/O) port requirements) into the network. So, the nodes required either a flexible I/O port of their own which could be adapted to most unique interfacing situations or the nodes could be configured with a standard I/O port and the external device required to adapt its I/O ports to the nodes similar to what was done in the ARPA network (Ref 9:4-1). Node #4 must interface an external communication network into the local

communication network. To do this, it must have the capability to deal with the different link control protocols utilized on the different networks and also to resolve any information compatibility problems between the two networks. This compatibility involved such factors as information message structure and network code used.

From the above, three basic ideas evolved about the universal network interface device. First, the device should function similar to a store-and-forward concentrator with a message routing function. Secondly, the universal interface device might require a specialized I/O port to handle unique interfacing requirements; and lastly, it should possess the capability to handle two network link control protocols. Given these attributes, the universal network interface would satisfy the different interfacing applications of the network diagrammed in Figure 1-1.

Structured Analysis Activity Model

The previous paragraphs described some general concepts about the universal network interface device. The purpose of the SA activity model was to translate these concepts into requirements for the universal network interface device. An index to the model is provided in Figure 2-1 and can be used as an overview to the functions the system must perform.

A SA activity model consists of a series of diagrams which present in progressively more detail the activities

Node	Title
A-0	Universal Network Interface Device
A0	Universal Network Interface
A1	Process Local Information
A11	Receive Local to-be-Transmitted Information
A113	Store Information
A12	Process to-be-Transmitted Information
A13	Transmit Information to Network
A2	Process Network Information
A21	Receive Information from Network
A22	Process Information from Network
A23	Retransmit Network Information on Network
A24	Retransmit Network Information to Local Receiver
A243	Process Control Information
A244	Transmit Information to Local Subscriber

Fig. 2-1. SA Activity Model Index

necessary to perform some function. The SADT activity model begins with node A-0. This node serves as a cover sheet for the model; the node is simply a box showing inputs, outputs, controls, and mechanisms for the function which the model is to describe. The text describing node A-0 begins on page 15. From that point on in this chapter, the text for each node is on a separate page which faces the figure showing the node.

In addition to an activity model, the SADT requires a data model be developed. This model describes how the data is changed after being acted upon by a given function. During the design of the universal network interface device, a data model was prepared. Because of the limited data being acted upon, the data model did not reveal further insight into the requirements of the universal network interface device. Thus, it is not included in this paper.

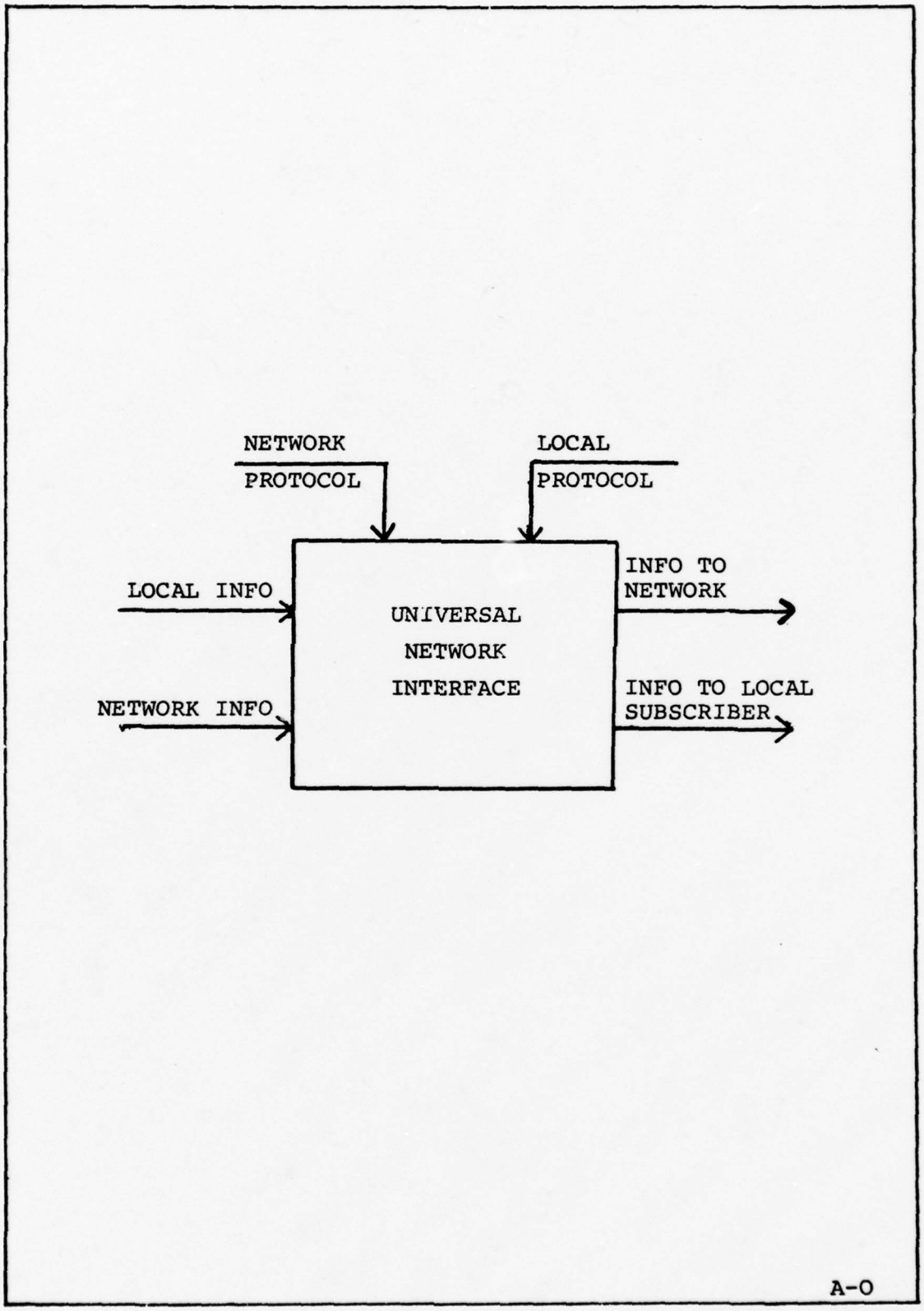


Fig. 2-2. Universal Network Interface Device

Universal Network Interface (A-0). Node A-0, (Figure 2-2), is the cover node for the SA model for the universal network interface. The purpose of the model is to define the functional requirements for the universal network interface device. The device receives data information bits either from local subscribers (i.e., peripherals) or from the network of which the interface is a component part. These information data bits are then processed by the network interface to determine the network addressee for the information data bits and the response required to satisfy the network's link control protocol or the peripheral's link control protocol. The information data bits are then transmitted either to a local subscriber or back onto the network along with any protocol-demanded response.

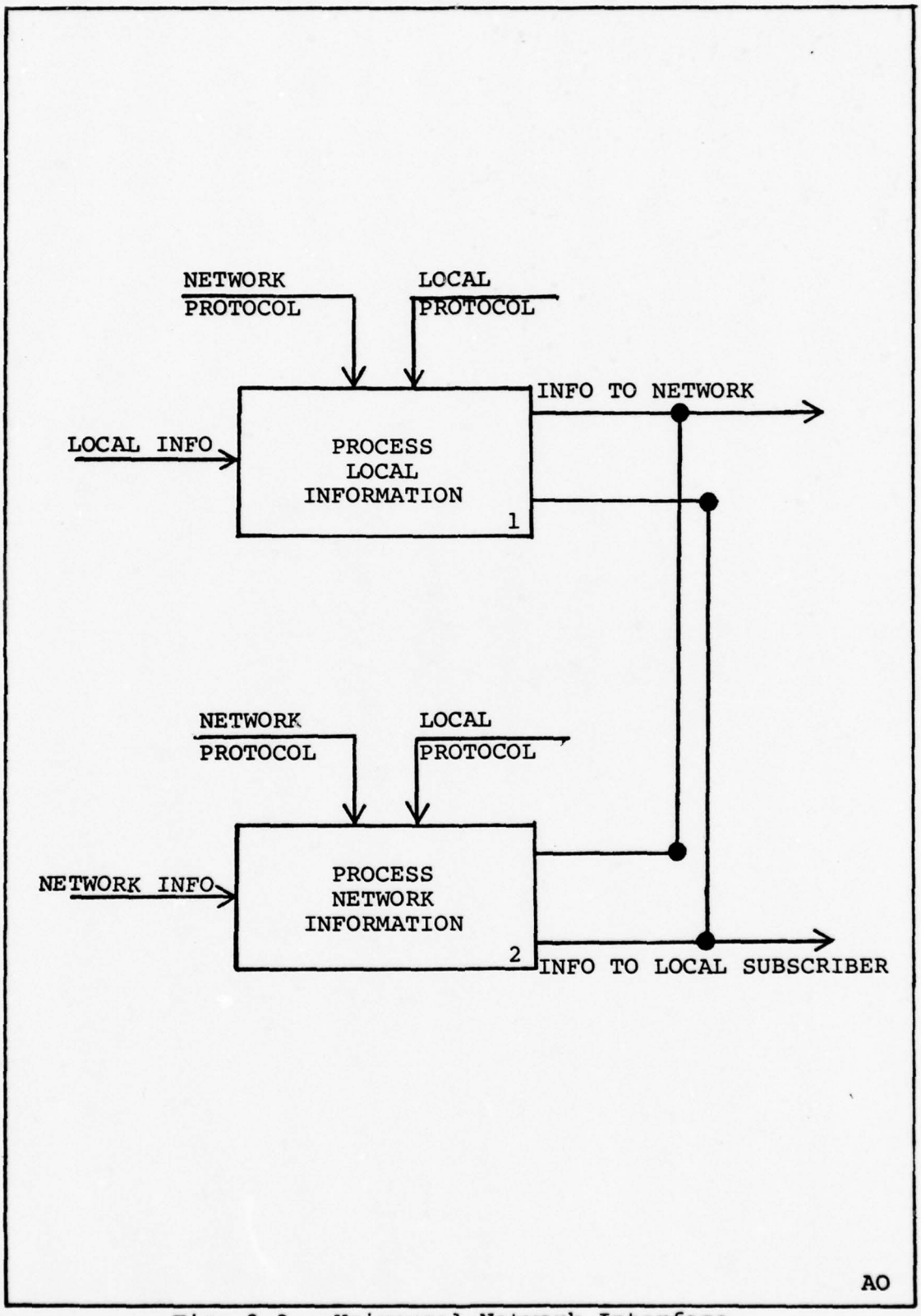


Fig. 2-3. Universal Network Interface

Universal Network Interface (A0). Node A0 in Figure 2-3 segregates the operation of the network interface into two functional processes: the local information process (1) and the network information process (2). Again, in both cases, data bits classified as local information or network information are acted upon by their respective processes and are then transmitted to the network and/or local subscriber. In the local information process, the local information is transmitted to the network and a response dictated by the peripheral link control protocol sent back to the peripheral. In the network information process, the destination of the information is determined. The information is then sent to a local subscriber or back to the network as a result of its destination address. Network link control protocol and peripheral link control protocol must also be transmitted.

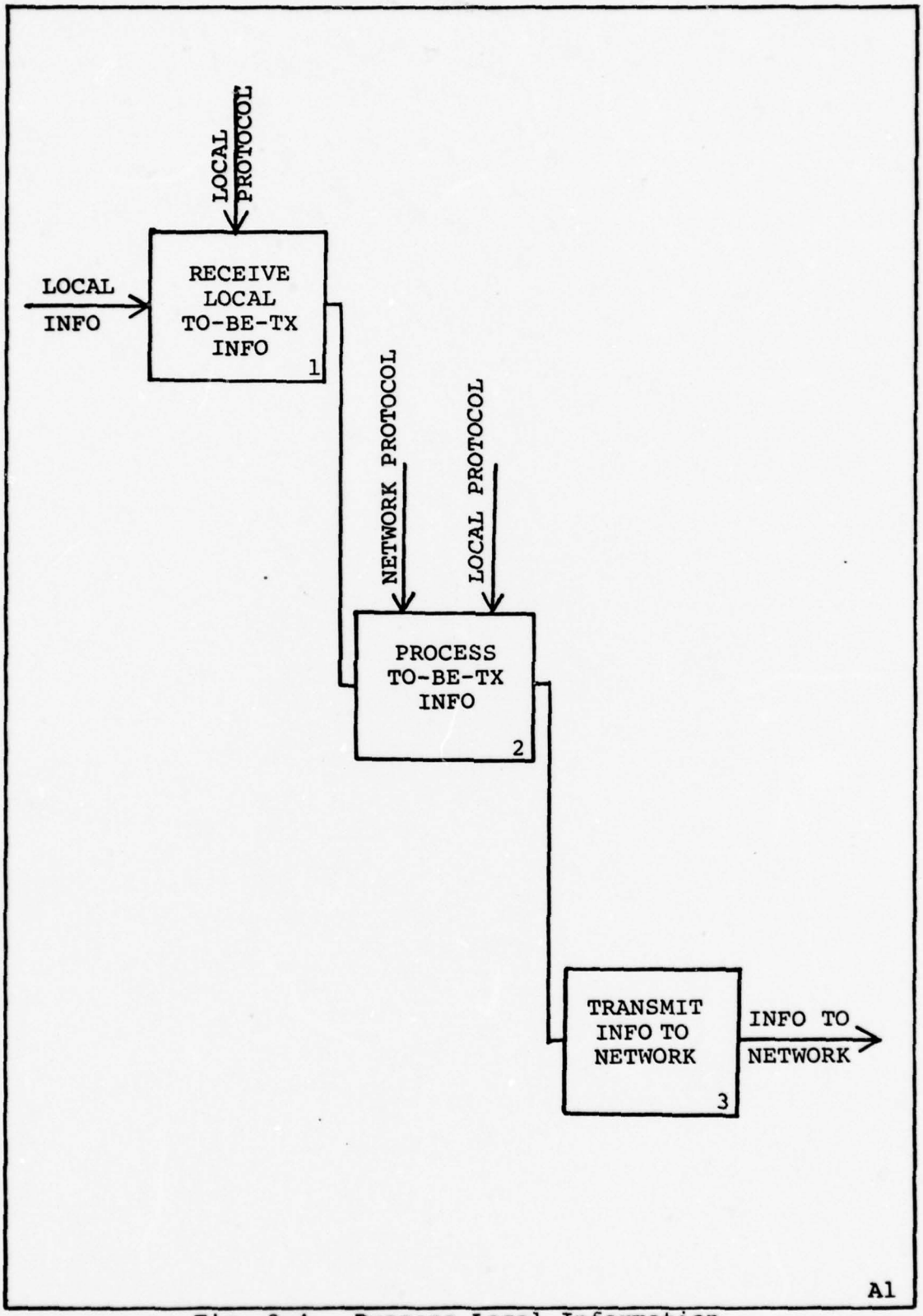


Fig. 2-4. Process Local Information

Process Local Information (A1). Process Local Information, Node A1, is presented in Figure 2-4. The function described in the diagram is the conversion of the local information bit stream into the format specified for the network message bit stream. To insure this conversion is transparent to the local subscriber, the local information is temporarily stored (1) within the network interface. The local information is then acted upon by the to-be-transmitted information process which formats the local information according to the network message format and the network link control message format. The local information is then transmitted on the network (3).

The function, receiver local to-be-transmitted information, also has a secondary usage of providing local storage for consolidation of character bits into information messages. In certain cases, the peripherals connected to the interface will not have enough local peripheral storage to develop a complete information message prior to sending the message to the network interface. The network interface through the receive local to-be-transmitted function should allocate storage space to the local peripheral to accomplish this consolidation.

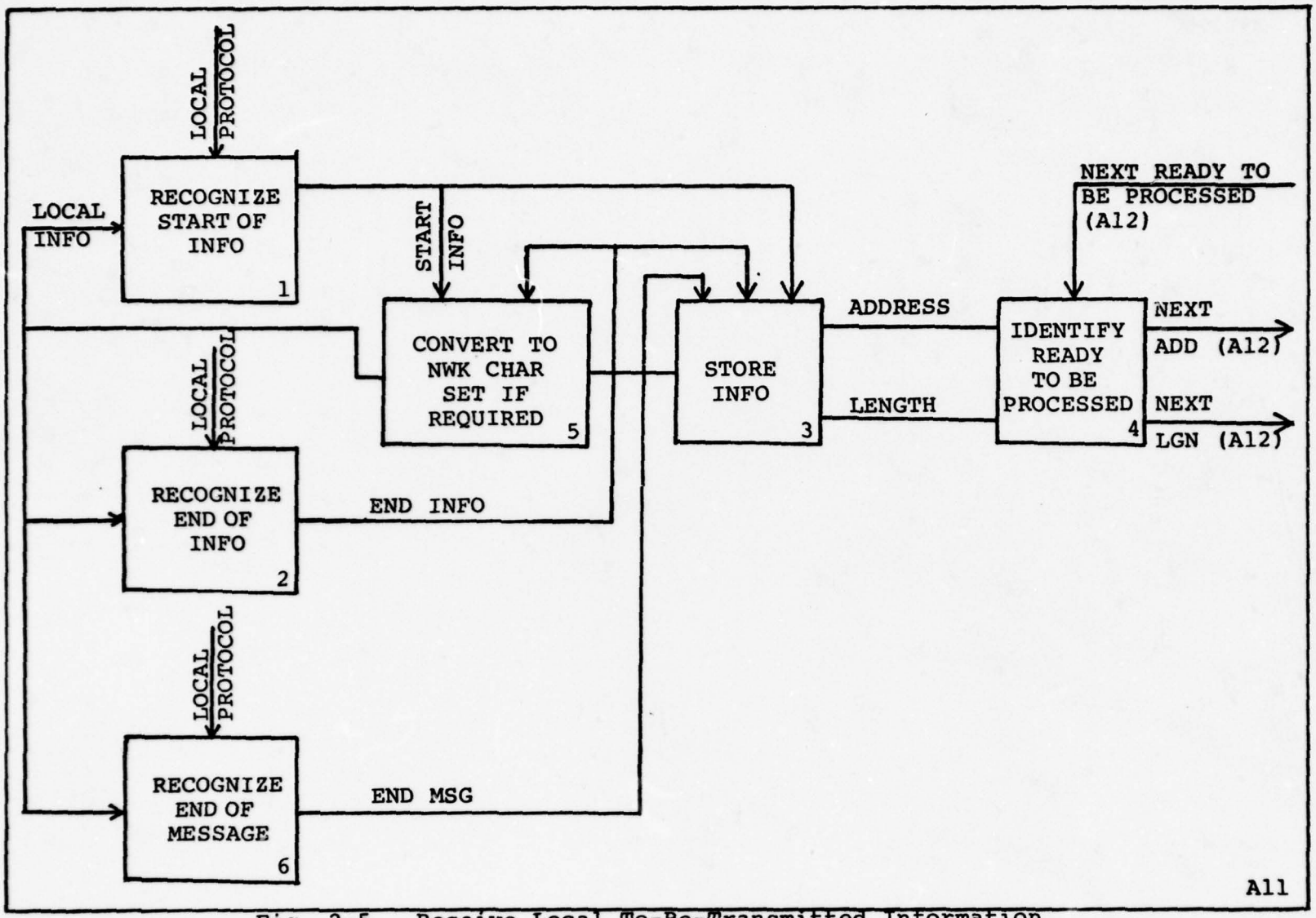


Fig. 2-5. Receive Local To-Be-Transmitted Information

Receive Local to-be-Transmitted Information (All).

The function of receiving local to-be-transmitted information is presented in Figure 2-5. The local peripheral communication line is monitored to ascertain the beginning of information. Once the beginning of information is detected, the characters within the bit stream must be converted to the standard network character set and then stored within the interface. The conversion is necessary to insure address information within the information bit stream can be interpreted by the other network devices. In addition, this conversion simplifies the interchange of information between two noncharacter compatible peripherals since only a local conversion between the network character set and peripheral character set is required. Incorporated into the store-information function is the need to break up the information bit stream into a number of subsets whose bit count is compatible with the storage medium size. Storage and conversion continues upon the local bit stream interrupted only by end-of-information characters. These end-of-information characters are established by the peripheral protocol to signal the interface to temporarily stop storing the information bits being received from the peripheral. This stopping and starting of information storage is finally terminated by an end-of-message character. The end-of-message should be a special character established by the peripheral protocol which signifies the message can now be transmitted on the network. Once the

storage of the message has been completed, the memory storage address and message length is provided to the identify-as-ready-to-be-processed (4) function. This function manages a list of the message memory addresses and message length of all local messages requiring processing. Messages are added to the list by the store-information function and removed from the list by the identify-as-ready-to-be-transmitted function.

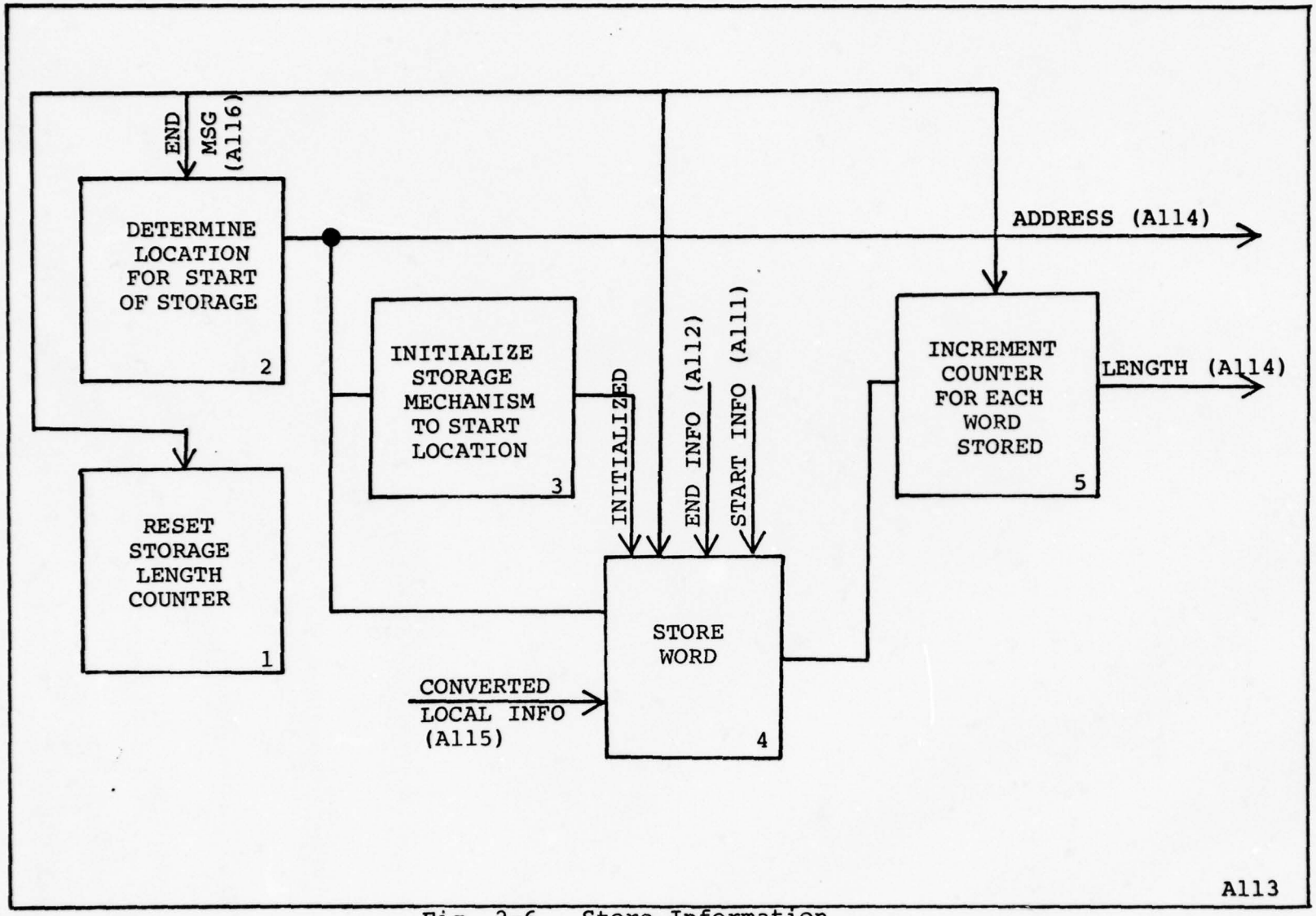


Fig. 2-6. Store Information

Store Information (A113). Figure 2-6 shows the functions necessary to store information. The determine-location-for-start-of-storage (2) function determines what memory is available. A memory block is reserved for this use and the start address of the memory block is used to initialize the storage mechanism. The store-word function accomplishes the actual storage of the information word. This storage will not take place unless a start of information has been detected and the storage mechanism has been properly initialized. As the local information data bits are stored, an increment counter response is also accomplished to accumulate the total storage length. This storage and increment process continues interrupted only by end-of-information characters until an end-of-message character is detected. At this time, the storage address and storage length are provided to the identify-as-ready-to-be-processed function. The store-information function is then reinitialized in anticipation of the next start storage character.

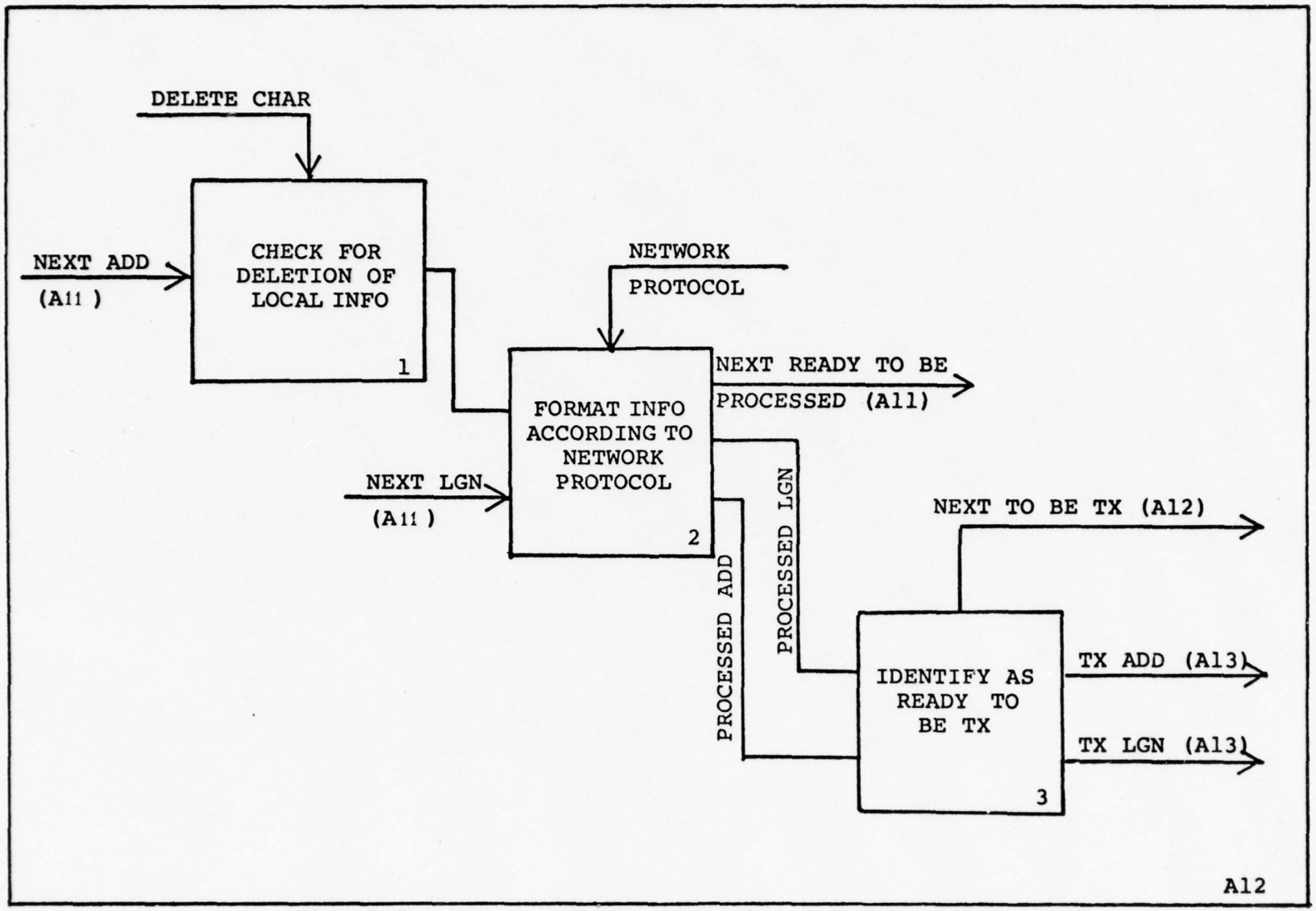


Fig. 2-7. Process To-Be-Transmitted Information

Process to-be-Transmitted Information (A12). The functions of the process to-be-transmitted information are diagrammed in Figure 2-7. The local information data words are checked for special characters signifying deletion and correction of previously provided data bits. These changes are made by the function and the corrected information sent to the format-message-according-to-network-protocol (2) function. This function then formats the information according to the network message structure in use, adds any network link control protocol-specified data bits to the local information and then identifies this total information block as a ready-to-be-transmitted network message. The memory address and memory length is then stored by the identify-as-ready-to-be-transmitted (3) function. This function provides a central storage point for all messages ready to be transmitted. Messages are added to the storage point by the format-information-according-to-network-protocol function and are removed from the list by the transmit-information-to-network function.

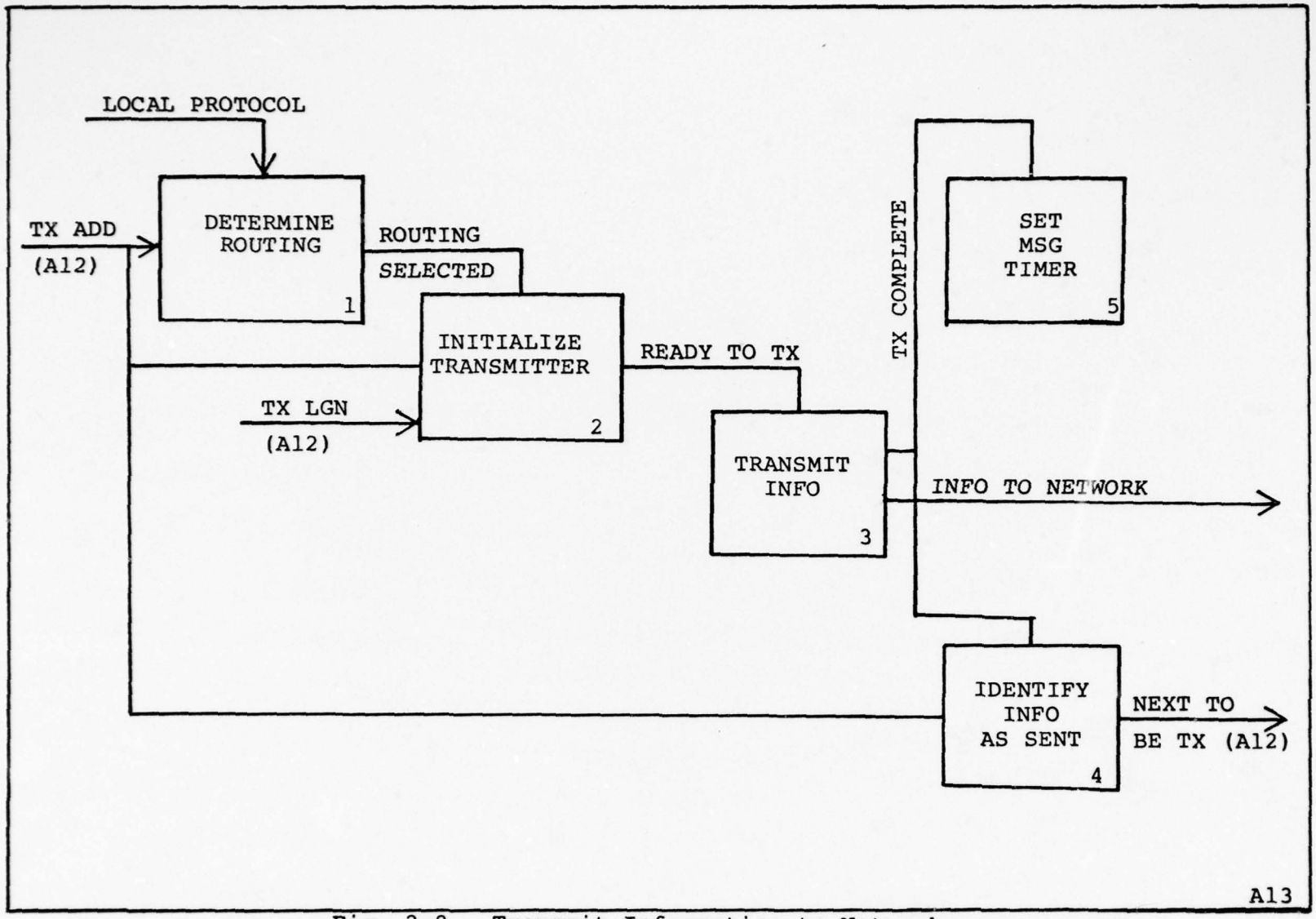


Fig. 2-8. Transmit Information to Network

Transmit Information to Network (A13). Node A13, shown in Figure 2-8, accomplishes the actual transmission of a network message. The transmit-information-to-network function first provides the local memory storage address of the next ready-to-be-transmitted message. The destination address of the message is then used by the determine-routing function (1) to ascertain which network link the message must be transmitted over. The transmission device for that link is then initialized with the memory address of the message and the message length and the properly formatted message transmitted. Unspecified but possibly necessary is the need to calculate and then transmit at the end of the message an error control word as specified by the link control protocol in use. Once the message has been completely transmitted, it is identified as such by the identify-information-as-sent (4) function and is saved to await any acknowledgement process. The function then requests the next message address and message length from the identify-as-ready-to-be-transmitted function. In addition, a timer is set to insure an acknowledgement is received in a specified time period. If not, the message must be retransmitted on the network.

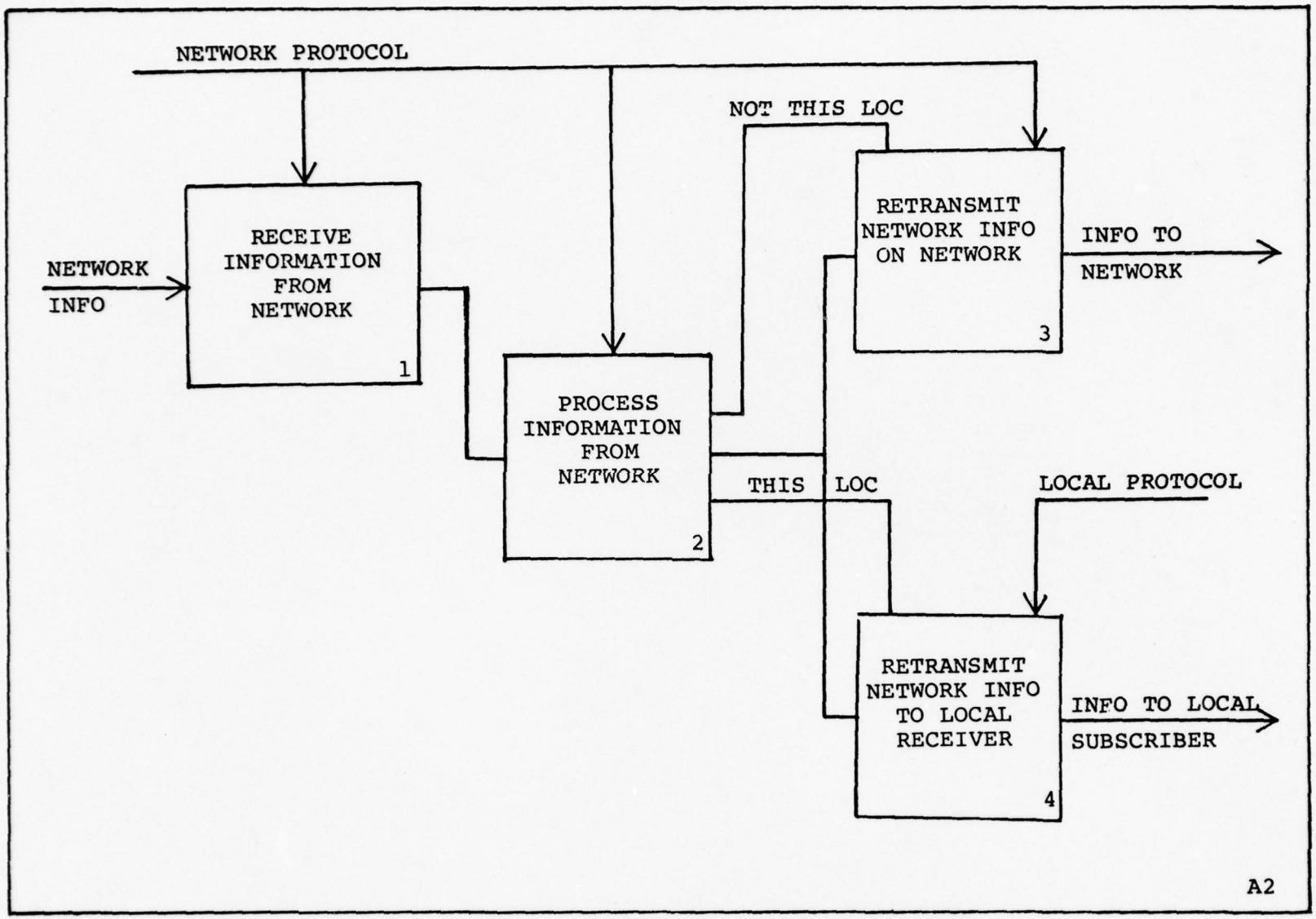


Fig. 2-9. Process Network Information

Process Network Information (A2). Process Network Information, Node A2, is presented in Figure 2-9. This node is the highest level in the second functional processes as defined by node A0. The functions described in the diagram include the reception of a network message and the resultant retransmission of the network message either to a local subscriber or back onto the network. The network information bit stream is first detected and stored by the receive-information-from-network (1) function. The received information bit stream is then processed by the process-information-from-network (2) function to ascertain the addressee of the message and whether the addressee corresponds to a local subscriber. The message is then sent to the retransmit network information on network (3) or retransmit network information to local receiver (4) depending upon the result of the addressee check.

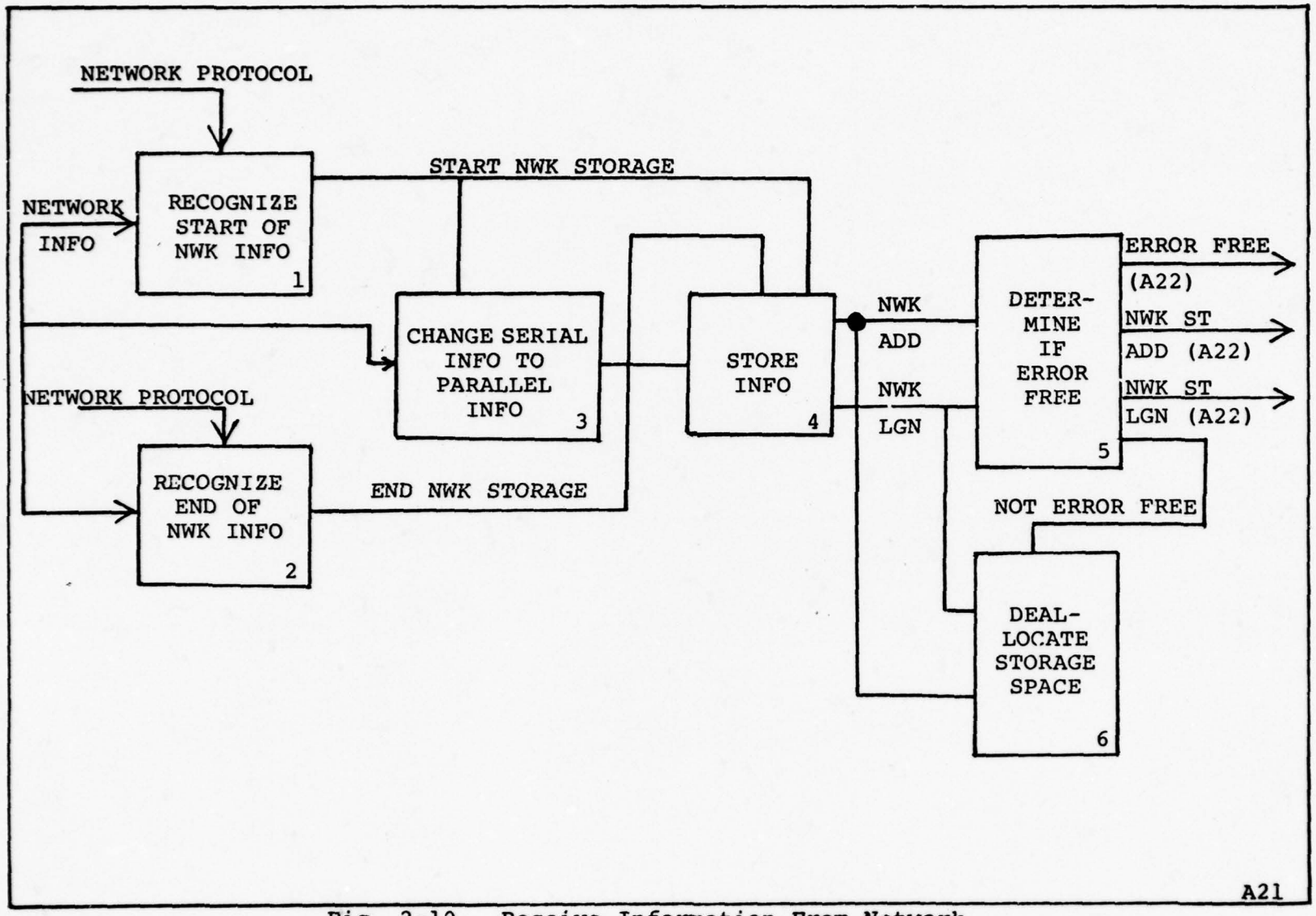


Fig. 2-10. Receive Information From Network

Receive Information from Network (A21). The function of receive information from network is presented in Figure 2-10. Any valid network information bit stream is detected by the recognize-start-of-information (1) function. This recognition process is controlled by the link control protocol. This protocol would stipulate the characters which would delimit the start and end of the message. Upon recognition of this special character, the change-serial-information-to-parallel-information (3) function accomplishes serial-to-parallel conversion to facilitate more efficient network interface storage of the network bit stream. Storage and conversion of the bit streams continues until an end of message is detected. This end of message would be a special character dictated by the network's link control protocol. This special character is detected by the recognize-end-of-network-information (2) function which in turn deactivates the conversion and storage functions. The store-information (4) function is identical in operation to the previous store-information function (A113) and will not be diagrammed at a lower level. The only difference between the two would be the information provided by the store function. In the latter case, a network storage address and network storage length are the outputs of the store function. Once the message has been completely received and stored, the message error word is checked by the determine-if-error-free (5) function to determine if the message was received correctly. If so, the network

storage address and length is sent to node A22 for further network interface processing. If not, the deallocate-storage-space (6) function is activated and the message deleted from the network's interface memory.

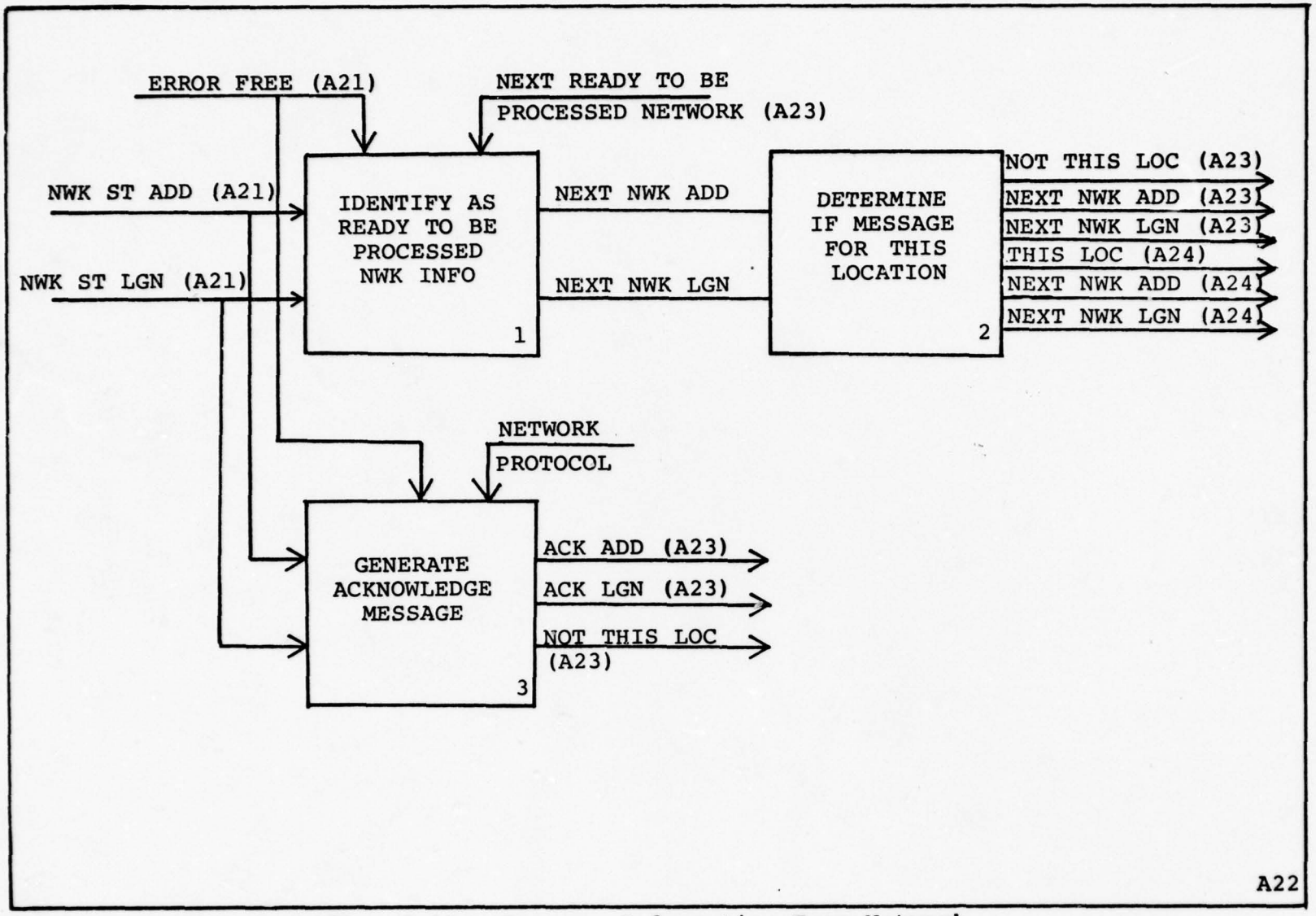


Fig. 2-11. Process Information From Network

Process Information From Network (A22). The functions of the process-information-from-network function are diagrammed in Figure 2-11. The identify-as-ready-to-be-processed-network-information (1) function acts as a centralized storage point for all correctly received network messages. Messages are added to the storage area if they are received error-free. Messages are deleted from the storage point by the retransmit-network-information-on-network function. Upon deletion, the memory address of a network message and its storage length are provided to the determine-if-message-for-this-location (2) function. This function determines if the message corresponds to the network address of any of the local subscribers. If so, the address and length is provided to node A24. If not, the same information is provided to node A23.

In addition, an acknowledgement message is generated to signify the correct reception of the message. The format for this acknowledgement would be dictated by the link control protocol being used.

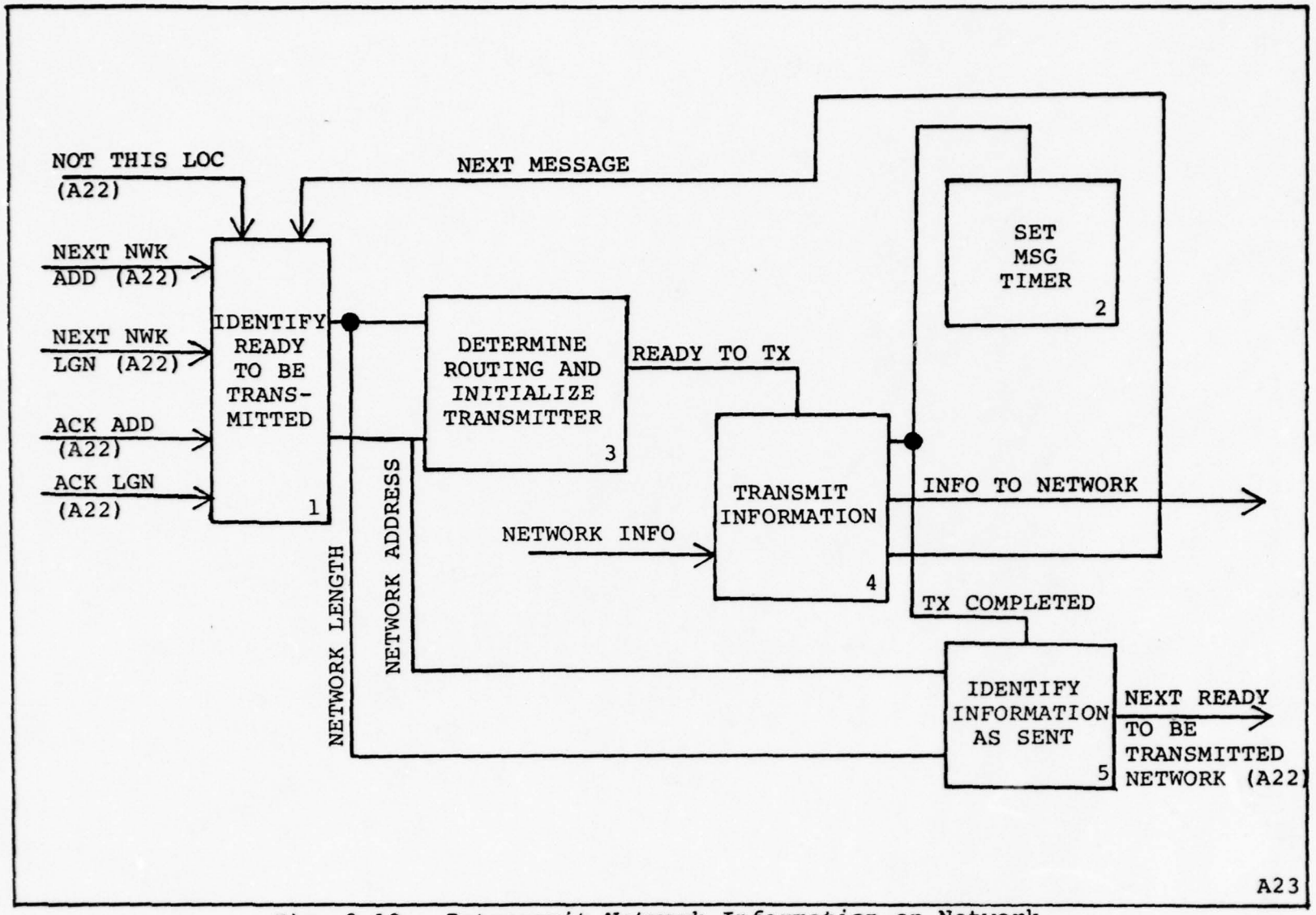


Fig. 2-12. Retransmit Network Information on Network

Retransmit Network Information on Network (A23). Node A23, which is shown in Figure 2-12, accomplishes the actual transmission of a network message. The identify-as-ready-to-be-transmitted (1) function acts as a central storage point for error-free network-received messages which must be retransmitted on the network. The address and length of those messages are stored under control of the not-for-this-location (A22) function. An address and length of a message is deleted from this central storage point by the transmit-information (4) function. Once the network address TX and the network length TX are sent by the identify-as-ready-to-be-transmitted function to the determine-routing (3) function, the operation on the address and length data is identical to that accomplished in node A13.

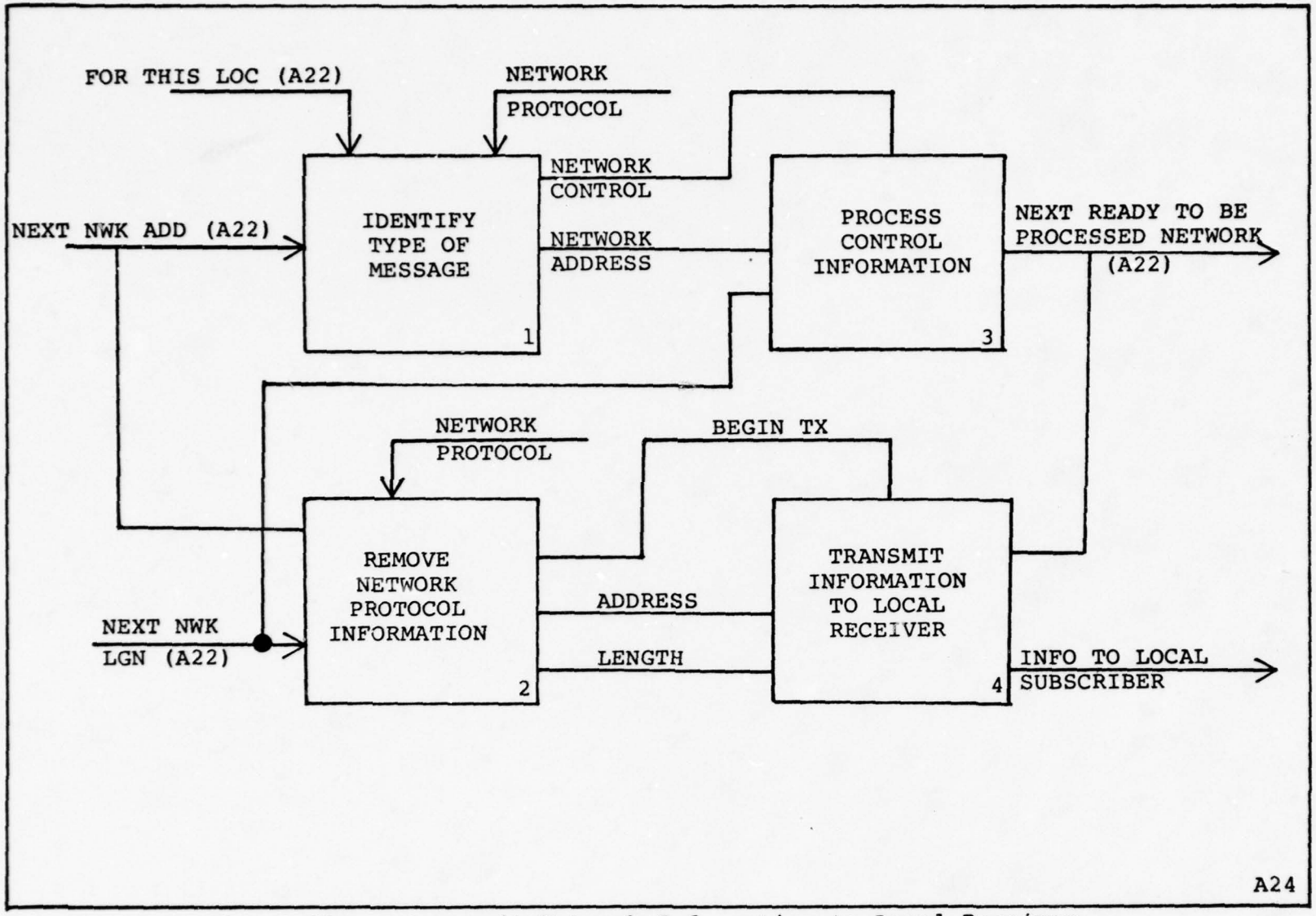


Fig. 2-13. Retransmit Network Information to Local Receiver

Retransmit Information to Local Receiver (A24).

Figure 2-13 shows the function retransmit information to local receiver. The identify-type-of-message (1) function receives the storage address of an error-free local message. It ascertains the type of message received. This type of classification then determines if the message address and length is sent to the process-control-information function or to the remove-network-protocol (2) function or both. In the control function, the control portion of the message is interrupted by the network interface and appropriate responses accomplished. The number and types of responses required would be dependent upon the link control protocol in use. In the remove-network-protocol-information (2) functions, the different bits added to the information stream to provide successful transmissions are removed and the address and length provided to the transmit-information-to-local-receiver (4) function. This function transmits the information message to the local subscriber. Upon completion of both the process-control-information function and the local transmission function, a new message is requested from the identify-as-ready-to-be-processed-network-information function.

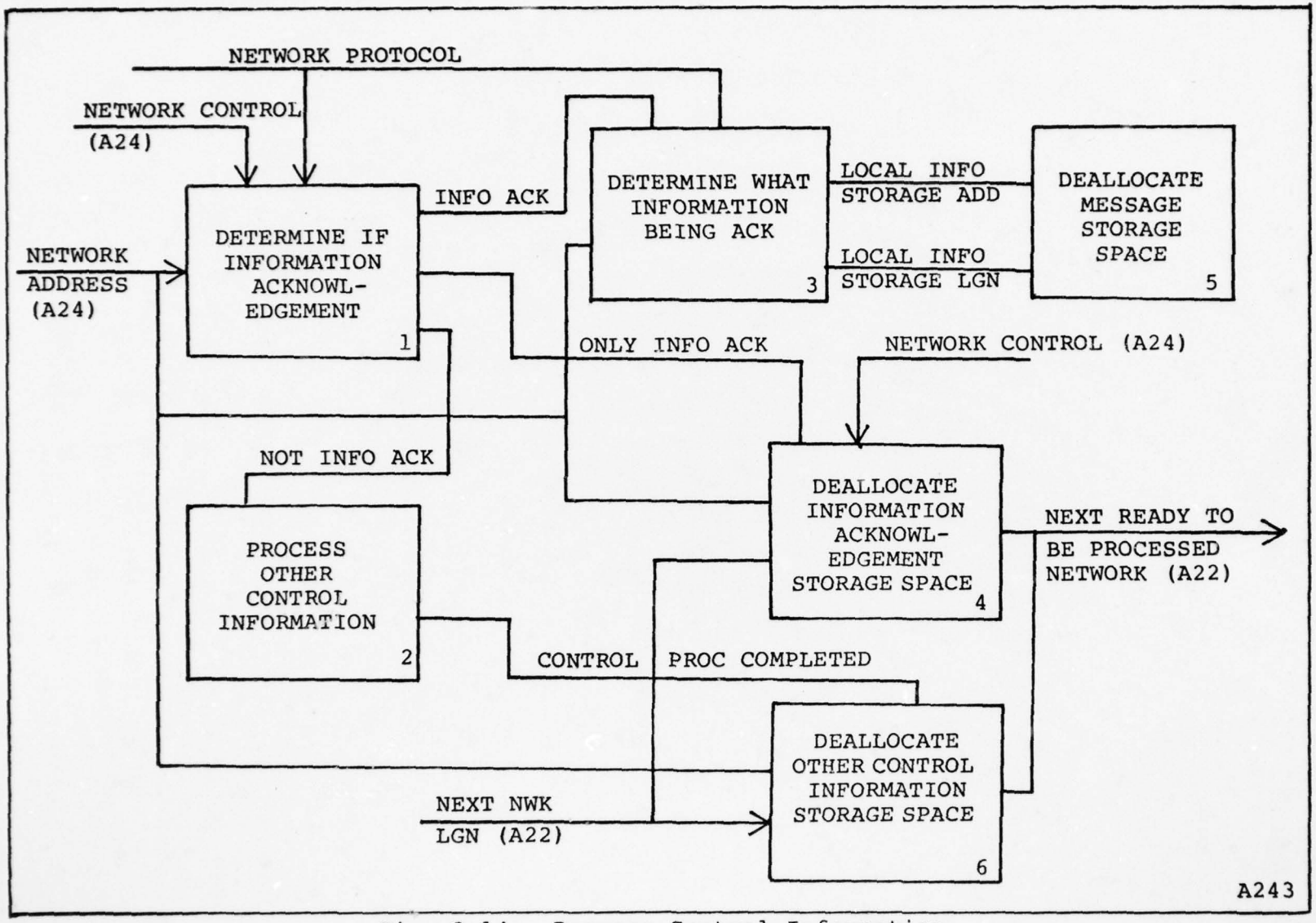


Fig. 2-14. Process Control Information

Process Control Information (A243). Figure 2-14 further breaks down the process control information function. The message address is used by the determine-if-information-acknowledgement function (1) and the network link control protocol message structure to determine if the message contained a message acknowledgement. If so, the address is sent to the determine-what-information-being-acknowledged (3) function. Here, the particular message being acknowledged is identified along with its storage address and storage length. This latter information is used by the deallocate-message-storage-space (5) function to return for use by other messages the previous message's storage space. If the received network message contained only an acknowledgement, the message address is provided to the deallocate-information-acknowledgement-storage-space (4) function which deallocates the message acknowledgement storage space. If the message did not contain an information acknowledgement, the message address is provided the process-other-control-information (2) function. This function determines the control information being sent and generates the appropriate interface response. If the message contained only control information, it is sent to the deallocate-other-control-information-storage-space (6) function which deallocates the storage space.

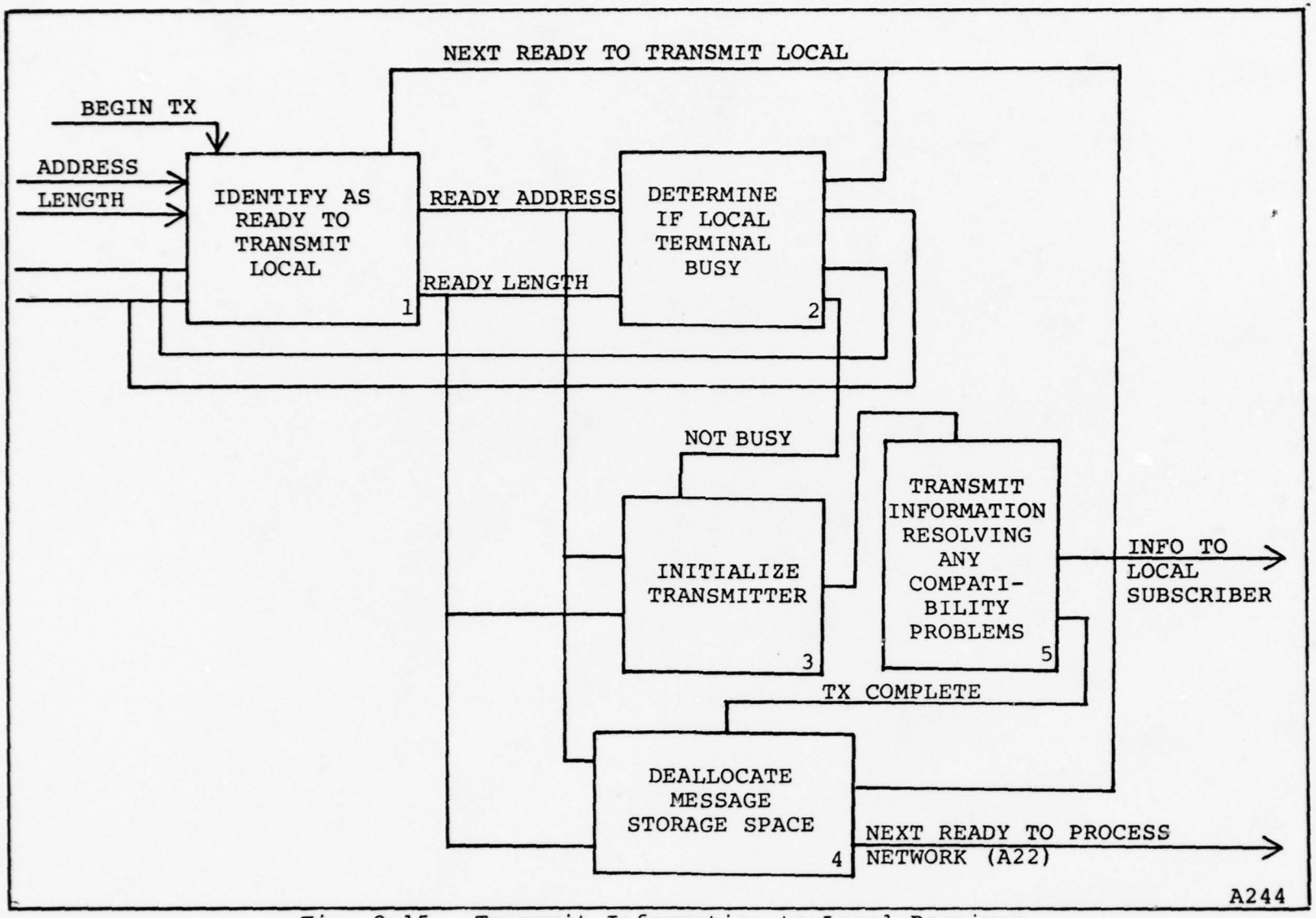


Fig. 2-15. Transmit Information to Local Receiver

Transmit Information to Local Receiver (A244). Node A244, Transmit Information to Local Receiver, which is shown in Figure 2-15, is the last node of the universal network interface. The identify-as-ready-to-transmit-local (1) function acts as a central storage point for all messages to be transmitted to local subscribers. It accepts message address and length information and provides a ready address and ready length to the determine-if-local-terminal-busy (2) function. This function determines if the local subscriber terminal is busy. If so, the ready address and length are returned to the central pool. If not, the initialize-transmitter (3) function is activated. This function sets up the transmitter for local message transmission. The message is transmitted by the transmit-information-resolving-any-compatibility-problems (5) function. This latter function is tasked to resolve any compatibility problems between the transmitted message peripheral and the received message peripheral. After the message has been transmitted to the local terminal, the message storage space is deallocated by the reallocate-message-storage-space (4) function.

Requirements Definition Summary

This concluded the requirements definition phase for the universal network interface device. This phase began with a concept of operation (Figure 1-1) for such a device. This concept was translated into generalized tasks the device must perform to satisfy the operational concept. Structure Analysis Design Techniques were then used to develop from the general tasks detailed functions for the device. The next step was to translate the individual functions into a design that would accomplish those functions.

III. System Design

The next phase of the design process involved the system design. In this phase, the functions identified in the requirements definition phase were allocated to hardware or software. However, before this allocation was accomplished, there were design uncertainties which had to be resolved. These design uncertainties are discussed in the first part of the chapter. Given these uncertainties, a method was devised to minimize the impact of the uncertainties on the design. The method used and the application of the method to certain requirement definition functions are then discussed. The last sections of the chapter discuss the hardware/software allocation and the processor requirements.

Design Dilemma

At this point in the design, the universality of the network interface device must be considered. In the normal design process utilizing SADT, the requirements phase would have consisted of specifying exactly what functions a system must perform and what timing restrictions it must meet. At this level of design, the specifications that are of interest are system inputs, the processing of the inputs and the system outputs. These would have, in turn, identified whether a given function's timing requirements or speed

of operation could have been satisfied in hardware, software or a hardware/software combination. It would then be up to the system designer to make the appropriate choice for the given function and then proceed with the design of the function. This, however, was not the case for the universal network interface's SA diagrams. Although the SA diagrams provided a general idea of the functions which the universal network interface must accomplish, they lacked the detailed specifications needed to proceed with the design. In a normal design, these specifications would be provided by the ultimate users of the completed device. In the universal network interface device case, the only specifications provided were the facts that the device should be universal and the general concepts of one possible network application in which the universal network interface device could be used. There was not enough information to proceed with the design. What was needed was system input/output information about:

1. Number of peripheral connected to the universal network interface device.

2. The speed of operation, the type of operation and the frequency of operation for the peripherals.

3. The number of network lines connected to the universal network inferface devices.

4. The speed of operation, the type of operation and the link control protocol in use on the network lines.

However, at this point, a conflict arose. As the specifications for the universal network interface device became more specific, the universality of the device decreased since the device became tailored to those specifications. If the requirements of the device were not defined more specifically, the design process could not continue. What was needed to resolve this, was some bounds on the requirements of the system I/O functions. This bounding would allow the device some universality since it would operate over a range and the bounds would provide the needed information to proceed with the design.

System Bounds

The need for system bounds was based upon the need for system input/output information and I/O requirements. This needed information was generally specified on the SA diagrams as the local information input/output and the network information input/output. The network information had been further classified by node A21 as being a serial bit stream. This represented a logical bound if the cost of the additional communication channels necessary for reception of parallel data and the problems involved in synchronization of parallel transmissions are considered. It also seemed reasonable to assume that the network information was of a synchronous type. This would allow more information to be transferred over a fixed capacity communication channel since the start/stop bits associated

with asynchronous communications would be eliminated. The other important characteristic associated with network information I/O was the transmission bit rate. This would be dependent upon the modem and the communication channel being used. The network shown in Figure 1-1 is to be implemented using the base cable system as the communication channel with a projected transmission rate of 1.5 mb/s (Ref 1:165). Other limited distance (ten miles) private wire lease lines have bit rates of approximation 1 mb/s (Ref 10:25). Most of the commercial networks are implemented over a switched (dial-up) or leased (dedicated) communication channel using the facilities of the common caririers. These carriers normally can provide voice channels capable of operating at up to 9.6 kb/s, half groups or full groups at 19.2 kb/s and 50 kb/s respectively, or even super groups at 230.4 kb/s (Ref 10:25). The transmission rates which could possibly be encountered in a network application range from approximately 1.2 kb/s to 1.5 mb/s with the 1.5 mb/s establishing the upper bound. The design bounds for the network information then became a synchronous, serial data stream of 1.5 mb/s.

Local Signal Characteristics. The local information characteristics were not further bounded by the SA diagrams. To develop these signal characteristics, the peripherals which were the source/recipient of the signals were examined. Datapro (Ref 11:222-239) categorized the data

communication peripherals into six major categories: (1) CRT terminals, (2) teleprinter terminals, (3) batch terminals, (4) cluster terminals, (5) intelligent terminals, and (6) special terminals, i.e., optical character readers. A review using Datapro of the different characteristics of these terminals revealed that approximately 90 percent incorporated an RS-232C data terminal interface into the terminal.

The RS-232C specification (Ref 12) establishes the interface requirements between data terminal equipment (DTE) and data communication equipment (DCE). This standard encompasses data interchange and control circuits, electrical voltage levels, impedance, transmission speed, slew rate and distance between the DTE and the DCE. As such, the RS-232C provides a good bound on the signal characteristics of the local information.

There was, however, one technical drawback to using the RS-232C interface in the universal network interface device. Any device utilized within a military system must meet the applicable military standards which, in this case, were MIL-STD-188-114 (Ref 14). These standards required a slightly modified RS-422 or RS-423 interface be employed between DTE and DCE. In a very strict sense, these standards should be utilized for the universal network interface device. However, given the fact that the majority of terminals utilized an RS-232C interface, it seemed more efficient to utilize this specification for the

interface. As the RS-422/RS-423 specifications begin to be employed in the design of new terminals, the universal network interface can be modified to incorporate these standards or the RS-XYZ interface (Ref 15) can be used to allow an RS-422/RS-423 terminal to be interfaced into the universal network interface device.

One additional aspect must be considered for the local information bounds. Many of the terminals in use today in the military environment employ a current loop configuration for transmission/reception of information. A 20 ma current loop interface would be a useful capability to include in the universal network inferface device. This would allow easy interfacing of those terminals which employ a current loop arrangement. For this reason a 20 ma current capability was established as a secondary bound.

I/O Port Requirements. The previous paragraphs established certain bounds on the I/O signals for the universal network interface device. Once the characteristics of these signals had been developed, the next aspect which was considered was the I/O requirements. The I/O requirements should specify the number of I/O ports the universal network interface device must accommodate. The SA diagram reflected a single local information input/output and a single network information input/output. While these two inputs/outputs were all that were required to develop the functional aspects of the device, these two inputs/outputs

would in most cases not meet the interfacing requirements of a given network. The two I/O requirements must be expanded and bounded to provide some universality and to also provide design requirements.

Network I/O Port Requirements. The first area considered was the network I/O requirements. The different topologies of a data network can be classified into three types--centralized, decentralized and distributed (Ref 16). The centralized network, (Figure 3-1), essentially a star configuration (links radiating from a single node), is the simplest arrangement. If the universal network device was employed at the end of a dedicated link to accomplish the concentrator functions, then the network I/O port requirement would be one full duplex port. A decentralized network, (Figure 3-2), is an expanded centralized network where the switching function, unlike the star arrangement, may occur at more than one location. In this arrangement, the universal network interface device would be employed between the switching function and the peripherals, thus again requiring one network I/O port. The distributed network consists of a set of mesh subnetworks in which each node of the subnetwork is connected to at least two other nodes. The individual rings in Figure 1-1 represent the simplest case. If one universal network interface device was employed as a concentrator for a subnet, the I/O port requirements would be dependent upon the number of subnets

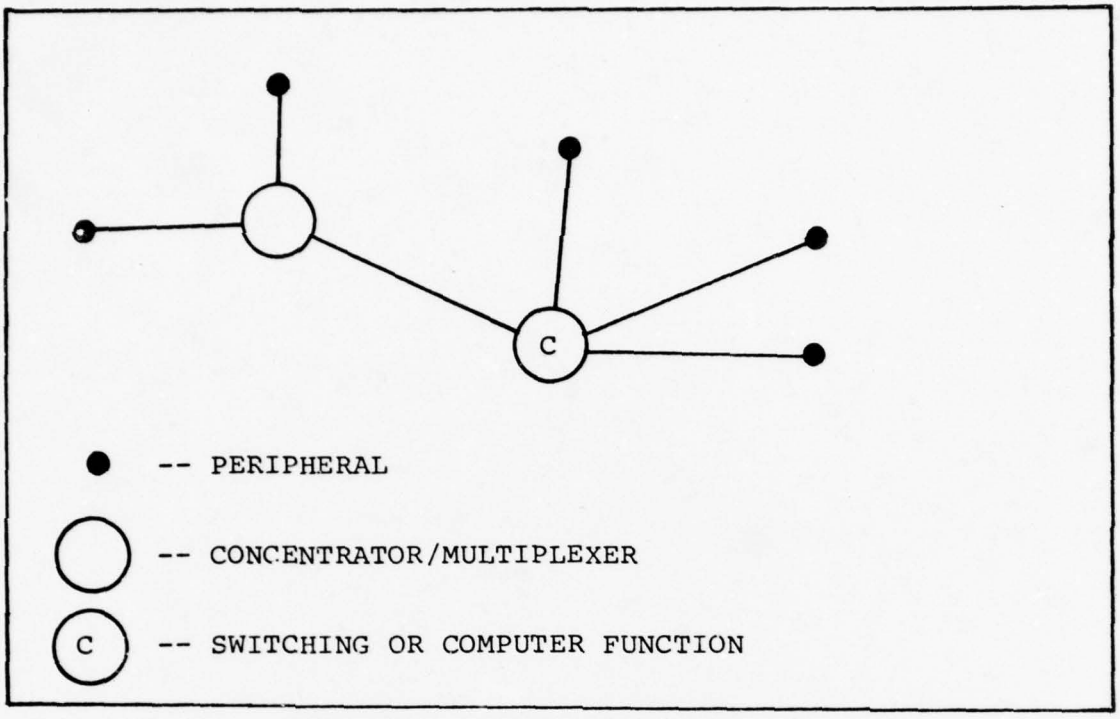


Fig. 3-1. Centralized Network

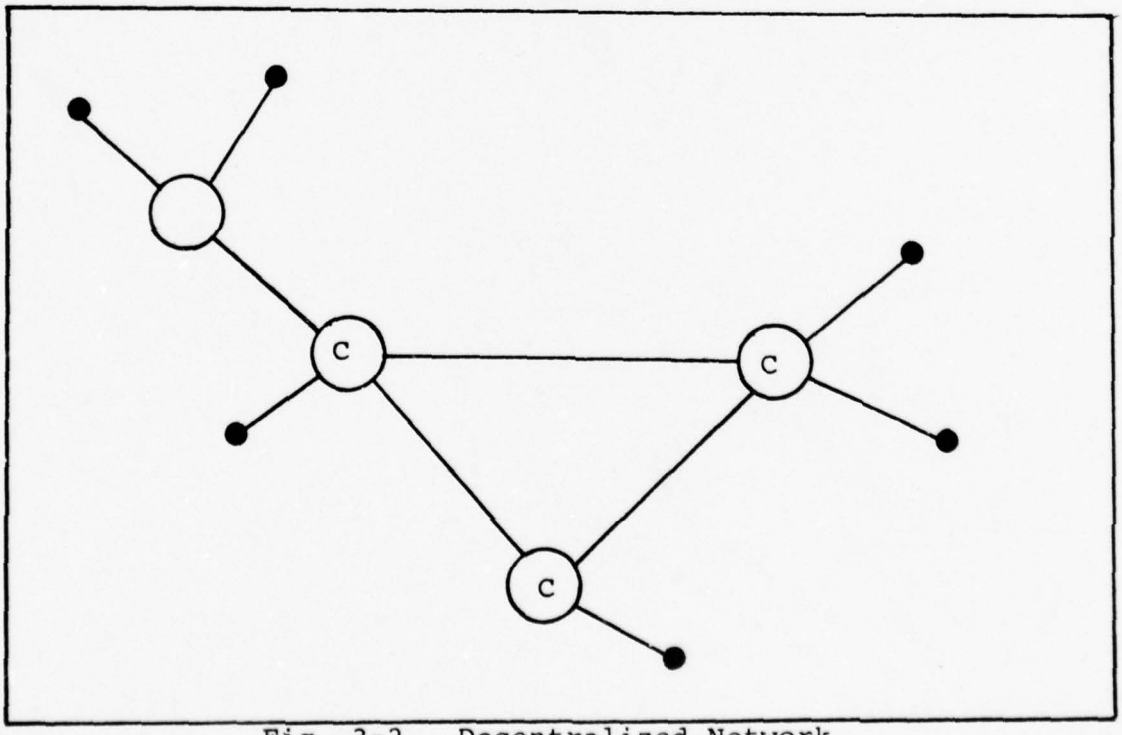


Fig. 3-2. Decentralized Network

and the interconnection desired between subnets. Figure 3-3 shows a simple distributed network where three I/O ports are required. One could continue to expand this arrangement generating more and more subnets and thus more I/O port requirements. Thus, there was no upper bound on the network I/O port requirement. The obvious thing to do at this point was to pick an arbitrary number and utilize this number in the design. However, the previous network evaluation suggested an alternative approach. From the previous information, the number of I/O ports varied; in one case one I/O port was required, in another case three ports were required and in a third situation an unknown number were required. This suggested the I/O port components of the universal network interface device be isolated from the other components. If the network I/O port was constructed on an individual card segregated from the other components of the device, the number of I/O ports could be expanded to meet the network topology requirements through incorporation of additional network I/O port cards. Thus, an upper bound did not have to be established from a physical point of view. The upper bound for the design could be established at a minimum figure of one network I/O full duplex port.

Local I/O Port Requirements. Now that the network I/O port requirements had been established, the local I/O ports were considered. Schwartz (Ref 17:136) listed two

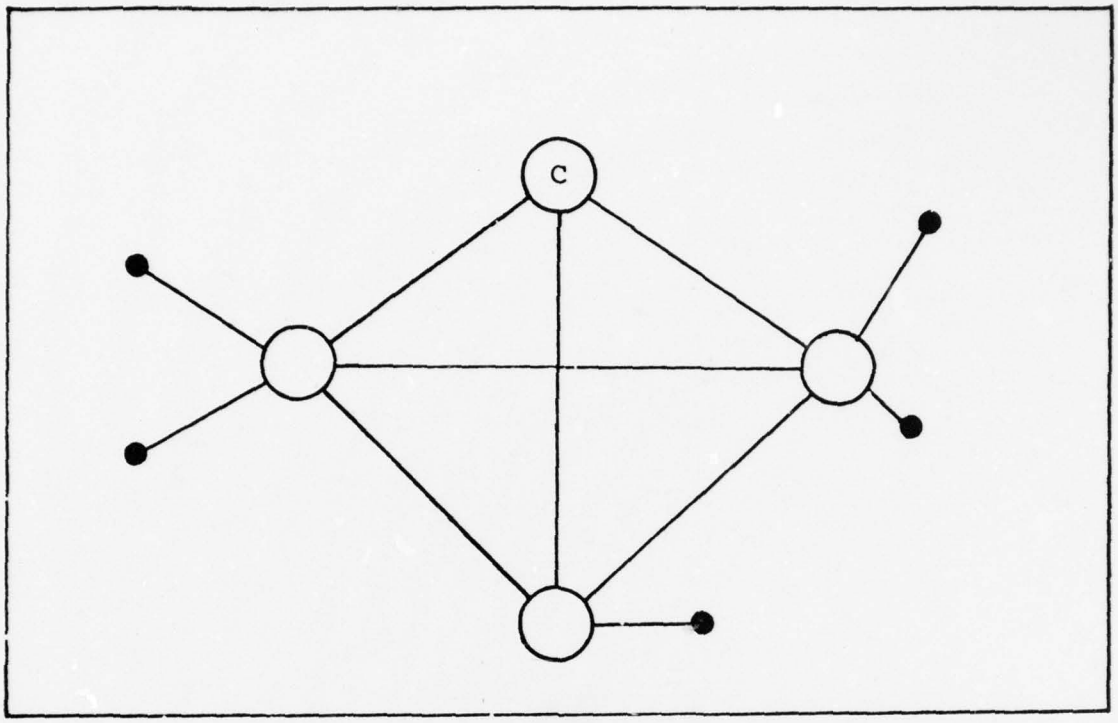


Fig. 3-3. Distributed Network

basic methods for entry of information into a concentrator-type device. Entry may be carried out by a scanning process (either sequentially or with priority) in which the various ports are continually scanned following a predetermined strategy to see if information is waiting to enter the system, or an interrupt procedure may be used in which incoming information notifies the system that it desires entry. In both of these cases, the local I/O port requirements become a function of the total amount of time dedicated to the entry function and the amount of time required to input/output the basic unit of information. For the polling sequence case, the average input/output time was shown to be (Ref 17:276) a function of the walk time (the time

required to scan a port), the number of ports to be polled and the effective traffic intensity. What was important about this was the fact that each of the ports have an associated polling overhead time which increased the total amount of time needed to input/output the basic unit of information. Thus for a fixed amount of time, the polling technique can service a lesser number of ports than the interrupt technique. Incorporated into this assertion was the assumption that the interrupt overhead time was less than the polling time. The upper bound for the I/O port requirements was thus established by the interrupt entry technique.

At this point, the number of I/O ports could be calculated if the total amount of entry time and the time required to service an interrupt for a given port were known. However, a situation was encountered which was similar to what occurred in the network I/O case. The upper bound could not be established since the service time and entry time necessary to calculate the upper bound had not been determined.

A complicating factor which affected this calculation was that the number of ports the universal network interface device can accommodate was a function of the characteristics of the terminals connected to the ports. The number of I/O ports established a situation which could be viewed as a classic single server queue (Ref 8:421). At any given point in time, a number of ports would be requesting

universal network interface service. All these requests formed the queue to the universal network interface device which acted on these requests one at a time. Thus, associated with any port's request for service was a queueing delay. Suppose one of the terminals was an unbuffered type and this terminal generated a request for service to input a character byte of information. If the queueing delay was long enough, the character byte of information would be changed prior to the previous character bit service request reaching the server. This latter situation would be unacceptable and the universal network interface device should not be employed in such a situation.

Thus, even though at a certain stage in the design process, an upper bound may be calculated on the number of local I/O ports, there is no assurance given the queueing delays and the service requirements of the individual terminals that service could be provided to a percentage of those terminals. This suggested an approach identical to the network I/O port requirements. The local I/O port functions should be designed on a separate card with a minimum number of ports per card. Since these ports must meet the RS-232C interface standard, the card should contain an "optimum" number of RS-232C interfaces. These cards can then be used to configure the universal network interface device with the number of local I/O ports that can be provided service.

Link Control Protocol

Most of the functions specified in the SA diagrams were now specific enough to proceed with implementation. There was still one function, format-message-according-to-network prototol, which required expansion. A major part of the usefulness of the universal network interface device revolved around the device's ability to handle the different link control protocols in use today. The major commercial protocols are shown in Table I. These link control protocols are the ones most typically discussed in the newer communication books (Refs 17:328-338; 18:369-386) and represented a good lower bound for the link protocol requirements for the universal network interface device.

In the military environment, the most logical application for the universal network interface device would be as a terminal subscriber within the Automatic Digital Network (AUTODIN). The particulars of the AUTODIN system can be found in reference 19. Briefly, the AUTODIN network link control procedures are a character-oriented control procedure. These characters are used to frame a basic unit of information transfer called the line block. The line block consists of two link control characters, followed by 80 text characters, followed by a link control character and then the block parity character. The block parity character may be either odd or even in parity and is formed by the binary addition without carry (sum modulo 256) of all bytes in the line block. Any message greater than 80

TABLE I

PROTOCOL CHARACTERISTICS (Ref 10:62)

Feature	BISYNC	SDLC	ADCCP	HDLC
Full Duplex	No	Yes	Yes	Yes
Half Duplex	Yes	Yes	Yes	Yes
Serial	Yes	Yes	Yes	Yes
Parallel	No	No	No	No
Data Transparency	Character Stuffing	Bit Stuffing	Bit Stuffing	Bit Stuffing
Asynchronous Operation	No	No	No	No
Synchronous Operation	Yes	Yes	Yes	Yes
Point-to-Point	Yes	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes	Yes
Error Detection (CRC)	CRC-16	CRC-CCITT	CRC-CCITT	CRC-CCITT
Retransmit Error Recovery	Yes	Yes	Yes	Yes
Bootstrapping Capability	No	No	No	No

NOTES:

Binary Synchronous Communication (BISYNC)
Synchronous Data Link Control (SDLC)
Advanced Data Communication Control Procedure (ADCCP)
High Level Data Link Control (HDLC)

characters in length is broken into a number of line blocks for transmission. There are five different modes of operation with mode I being the most efficient. Mode I is full duplex, synchronous operation with automatic error and channel controls which allow independent and simultaneous two-way operation. The line control characters utilized within the AUTODIN system are identical to those of the BISYNC protocol.

The basic protocol requirements for the universal network interface device should thus include the popular commercial protocols and the AUTODIN protocol. This is not an exhaustive list of all the different link control protocols in use. However, the universal network interface device must at least accommodate the protocols identified. This implied most of the protocol functions would be done in software. To implement other unspecified link control protocols would involve only a software effort.

Function Allocation

The different functions identified in the SA diagrams had now been expanded and bounded to allow continuation of the design process. Once the requirements definition model had been constructed, it was decided some type of LSI processor was needed. This decision was based upon the universality aspect of the universal network interface device. While most of the functions could be implemented in a specialized hardware design, this design would have to

be changed as the network environment changed. The LSI processor approach allowed a more flexible universal network interface device to be designed by allowing changes to be made in software.

Several processor implementations seemed possible; among them, a single-board computer, a bit-slice microprocessor, a microprocessor with special-purpose hardware or a multiprocessor configuration. The problem was to determine which processor implementation would work and which one was most efficient. In addition, since the universal network interface device would incorporate a processor, the different functions identified on the SA diagrams had to be allocated to a hardware or software implementation.

The software/hardware allocation task was completed first. This involved the assignment of a given function either to the processor for accomplishment or to the input card or network card. These later two cards implemented the expandability concept developed previously. They incorporated those functions which were associated with I/O ports and which must be expanded to meet additional I/O port requirements. Tables II, III and IV provide the different breakouts of the functions. Note that all of the functions were not specified in one of the three tables. This was caused by the fact that if a higher level function was specified in software, the lower functional breakouts of the higher level function were not included in the tables.

TABLE II

INPUT CARD FUNCTION

Node	Title
A11	Recognize Start of Information
A12	Recognize End of Information
A16	Recognize End of Message
A244 3	Transmit Information

TABLE III

NETWORK CARD FUNCTION

Node	Title
A211	Recognize Start of Network Information
A212	Recognize End of Network Information
A213	Change Serial Information to Parallel Information
A234	Transmit Information

TABLE IV

SOFTWARE FUNCTION

Node	Title
A113	Store Information
A114	Convert to Network Character Set if Required
A115	Identify as Ready to be Processed
A12	Process to-be-Transmitted Information
A131	Determine Routing
A132	Initialize Transmitter
A134	Identify Information as Sent
A214	Store Information
A215	Determine if Error-Free
A216	Deallocate the Storage Space
A22	Process Information From Network
A231	Identify as Ready to be Transmitted
A232	Determine Routing
A233	Initialize Transmitter
A235	Identify Information as Sent
A241	Identify Type of Message
A242	Remove Network Protocol Information
A243	Process Control Information
A244*	Transmit Information to Local Receiver

NOTE:

*All of the Transmit Information to Local Receiver was not allocated to software. The Transmit Information portion of A2445 was allocated to the Input Card Function.

The breakout between the different cards and processor tended to be fairly easy. Both the input card and the network card were allocated the functions associated with recognizing the serial bit stream, converting this serial stream into a composite word and then having the processor store the word. What was envisioned here was for the different cards to construct a word of the same size as the word used by the processor and thus the processor would only have to store this word. To have the processor do any tasks on the serial bit stream would be inefficient. Likewise, for the transmit function the cards should accept a computer word and convert this into a serial bit stream for transmission. The only other function which might be allocated to the network card was the determine-if-error-free function. This function involved an arithmetic computation on the individual words within the message and the comparison of this calculated result to the error word at the end of the message. From a universality point of view, this function should be allocated to the processor since it would be able to accomplish any type of arithmetic computation specified by the error control techniques of the different link control protocols. However, if the processor does accomplish this, the effective storage speed per word will be reduced if this calculation is done as the word is received.

Processor Requirements. The functions in the requirement definition model which are allocated to software are shown in Table IV. These functions establish the requirements for the type and number of processors required for the universal network interface device. This section of the paper determines the number required, while the type of processor is discussed in the next chapter.

The number of processors required hinges upon the number of tasks which must be performed and the time limitation established for the performance of these tasks. One can again view this as a single server situation. In the universal network interface case, the peripherals connected to the interface transmit serial information to the device which is transformed into a word by the input or network card. The ports on the cards then request the processor to store the word. These storage requests plus the internal tasks form the queue to the server (processor). The server extracts the task from the queue and executes the given task. There is, however, a time limitation imposed on the server in the universal network interface case. The server must execute the task (store the word) for a given I/O port prior to the next request from the same I/O port. If this is not done, the word is lost from the network unless there is a technique in use to detect this condition.

Now that a concept had been developed on how the interface would operate, the interface's ability to meet this concept was examined. This examination was based upon a

worst case situation. It was assumed the universal network interface's I/O ports had the capability to store one word. In addition, it was assumed that at a given instant during the day all network ports and peripheral ports requested service. This latter assumption only has a very small probability for occurrence; however, the device must be able to handle the worst case. Previously, the input I/O ports were defined to be RS-232C interfaces. Thus, the maximum transmission rate allowed on any one port would be 19.2 kb/s (Ref 12:3). Using this figure and the assumption there is 8 bits/word, the time between requests for storage on any given active port is 416 usec. If the processor requires 40 usec to service each port, then ten 19.2 kb/s devices can be connected with a reasonable assurance all will receive proper service. If the network employed a 50 kb/s transmission rate, only six 19.2 kb/s devices as shown in Figure 3-4 can be accommodated. If the network employed a 200 kb/s transmission rate, no 19.2 kb/s devices could be serviced during reception or transmission of a network message. Thus, in this situation, the universal network device would not meet the RS-232C specification design goal.

One could continue to calculate using different combinations of numbers the different configurations the universal network interface device could or could not service. More important was the idea of how the device's universality could be extended. Given the conditions assumed, there

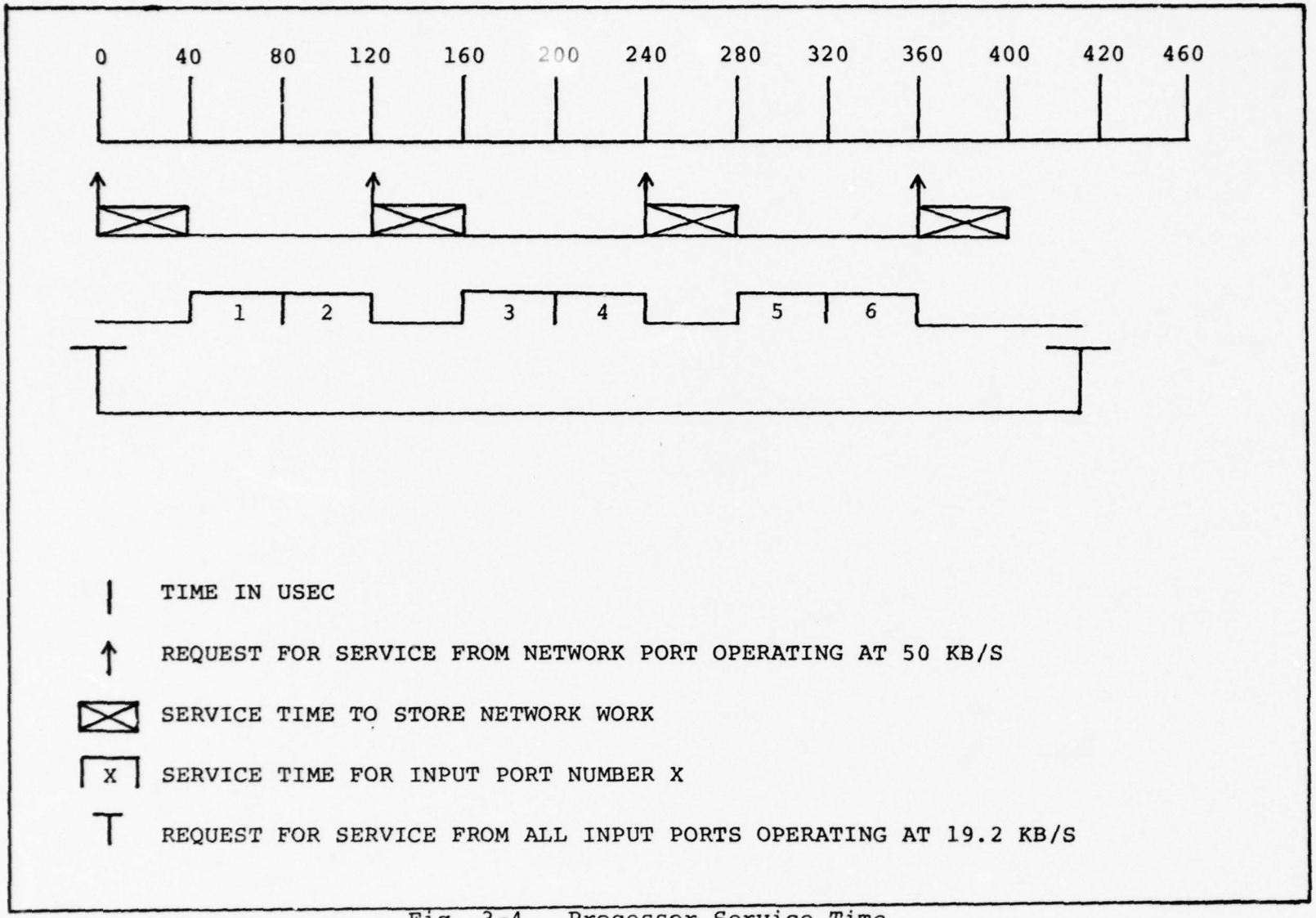


Fig. 3-4. Processor Service Time

was a situation where one processor could not provide the required service. However, there may or may not be an actual network with these parameters. Could these calculations then be the basis for a decision on the need for another processor? The ultimate decision should be made by the user of the universal network interface device based upon his required application. A way to accomplish this and extend the universality of the device was to revert again to a building block concept with regard to processors. The basic processing capability would be developed on one card and another card designed which would allow other processors to be added if required.

System Design Phase Observations

The system design phase started out to be a simple allocation task. However, the lack of specific functional requirements dictated these be developed by the system designer. What resulted was not only the specific requirements but a design philosophy. The result of this philosophy was a modular concept for the universal network interface. This modularity of functions thus allowed the device the degree of universality which had been hoped for at the beginning of the design process.

IV. Hardware Selection and Design

Once the first facet of the system design had been completed, the next phases involved the design of the hardware and the software. This chapter is concerned with the hardware aspects of the universal interface device. The chapter is divided into sections which correspond to the hardware component selection and design of the different cards--processor card, input card, network card, multiprocessor card--needed to implement the modularity concept.

Processor Selection

Two criteria were used to select a processor. The first was the capability to perform all the functions listed in Table IV. After functional capabilities, the next consideration was the simplicity of the processor. This latter factor becomes important, especially in a military environment, because of its direct relationship to life cycle cost. To minimize life cycle cost, the selection of the processor and associated hardware had to be such as to minimize the skill level necessary to perform hardware and software maintenance and modification. If the device was not designed to be cost effective over its life cycle, it would probably not be employed in a military environment.

There were two processing time considerations which affected processor selection. First, there was the general processing time of the processor. This impacted on three critical areas--the storage requirements, the throughput capability and the input/output design. As the instruction execution speed of the processor decreased, the service time to accept a completed message and transmit it increased. Thus, the throughput of the device decreased causing the local storage requirements to increase since, on the average, more messages must be stored locally. From a strictly throughput point of view, the processor chosen should have the fastest instruction execution time available. However, combined with this instruction time, an evaluation of the instruction sets had to be accomplished to insure the power of the instruction set did not offset an execution time advantage.

Previously, it had been established that the entry of information into the universal network interface device could be either through a polling technique or an interrupt technique. Once this serial data entered the interface by either technique, the input and network cards temporarily stored the serial bits to allow transformation into a parallel word. After the parallel word was formed, the cards requested the words to be stored. This request could be to the processor in the form of an interrupt request or it could be to a Direct Memory Access (DMA) device which accomplished storage without processor intervention. The

first type of request became important in the selection of a processor. The ability of the processor to handle an interrupt request with both minimum overhead and maximum efficiency was an important criterion used for processor selection.

Now that the criteria had been established, the different processors were considered. The criteria were applied to two processor options: a bit slice microprocessor and a conventional microprocessor LSI chip.

The bit slice microprocessor represented the logical choice if only execution speed was considered. A typical bit slice microprocessor had microinstruction execution times of from 100 to 200 nanoseconds (Ref 20:18-1) with program instruction execution times a multiple of the macro-instruction execution time. A bit slice microprocessor system, however, was more complicated than a conventional microprocessor. A microcontroller unit and a microprogram read only memory were required to determine the location of the next microinstruction and the microcode for that instruction. Since the bit slice processor instruction set was defined by the microcode which in turn had to be developed, software development and maintenance cost were greater than for a conventional microprocessor. This additional microcode software and added system complexity increased life cycle cost. The bit slice microprocessor approach was not selected based upon the life cycle criterion.

A conventional microprocessor implementation was used because it resulted in a good balance between execution speed and life cycle cost. To select the proper microprocessor, a benchmark code segment for the interrupt initiated word storage process was developed. The following sequence of instructions was considered the minimum necessary to accomplish such a task:

Store the working registers of the interrupted program
Determine the storage location for the word to be stored
Input the word into the processor
Store the word
Restore the working registers
Enable interrupts
Return to the main program

The benchmark was used to develop routines for the more popular microprocessors with general execution speed of 2 usec. The results of this comparison are shown in Table V. The instructions for the different microprocessors along with the number of clock cycles per instruction and the minimum clock cycle time was based upon information in reference 20.

From this evaluation, the three processors with the fastest execution time were selected as candidates and evaluated further. The RCA CDP 1802 was immediately eliminated since all interrupts caused the processor to begin executing instructions addressed by general purpose register R1. To differentiate between interrupts would require a number of branch-on-condition instruction that test the input flag (Ref 20:11-9), thus slowing interrupt processing. Of the two remaining, the Z80A was the better choice.

TABLE V

8 BIT MICROPROCESSORS CONSIDERED FOR THE UNIVERSAL NETWORK INTERFACE DEVICE

Microprocessor	Minimum Clock Cycle Time	Number of Clock Cycles for Benchmark	Total Time
Fairchild F8	500 nsec	44	22.0 usec
Intel 8085	320 nsec	120	38.4 usec
RCA CDP 1802	155 nsec	128	19.8 usec
Motorola 6800	1000 nsec	30	30.0 usec
TMS 9900	333 nsec	108	35.9 usec
Zilog Z80A	250 nsec	81	20.0 usec
Zilog Z80	400 nsec	81	32.4 usec

It had a faster execution time and its instruction set was more extensive than the F8's. As an example, the Z80A provided a single instruction to test an individual bit in a word which is stored in the registers or memory. The need for this bit testing tends to occur in most applications so a single instruction to test any bit becomes a very powerful tool. There were also single instructions to transfer blocks of data between two locations in memory and also between I/O ports and memory. This ability to transfer blocks of data seemed very useful for message transmission. The Z80A contained two sets of main registers thus allowing rapid processing of first-level interrupts. In addition, it was supported by a variety of support chips.

Processor Board. With the Z80A selected as the universal network interface device processor, the next step was to identify a Z80A microcomputer board which would meet

the interface requirements. The microcomputer board approach was selected to minimize the amount of uncertainty in the design. If the proper board could be identified, then design problems associated with memory interfacing, clock interfacing, etc. would be eliminated. The board selected was a Z80A-MCB developed by Zilog, Inc. Unfortunately, the board was still in development and would not be available until January 1979. However, the company manufactured a Z80-MCB which the Z80A-MCB was designed to replace. The Z80-MCB employed a Z80 processor with a clock of 403 usec. It was decided to utilize the Z80-MCB as the basis for the design. The design of the other cards was, however, based upon the clock rate of the Z80A.

Z80-MCB. The Zilog Z80-MCB is a single-board microcomputer card, the heart of which is the Z80 microprocessor. Associated logic includes 4K bytes of dynamic random access memory (RAM), provisions for up to 4K bytes of programmable read only memory (PROM), read only memory (ROM) or electronic programmable read only memory (EPROM), a parallel and a serial I/O port, an I/O port decoder and a crystal controlled clock. The parallel port is implemented with the Z80-PIO (parallel input output) chip. Also included on the board are four programmable band rate generators implemented through use of the Z80-CTC (counter timer circuit) chip. One band rate generator is used for the serial I/O port which is implemented with an Intel 8251

universal synchronous asynchronous receiver transmitter (USART). All address, data and control lines are buffered and feed to the 122-pin edge connector (Ref 22). Additional information on the Z80A/Z80 processors and the Z80-MCB is provided in references 20 and 22.

Input Card

Previously, it was determined the input card accomplished the functions of message and word recognition, serial to parallel conversion, and service request generation to the processor. The input card also met the RS-232C interface. Given these characteristics, the next step consisted of a design to meet them.

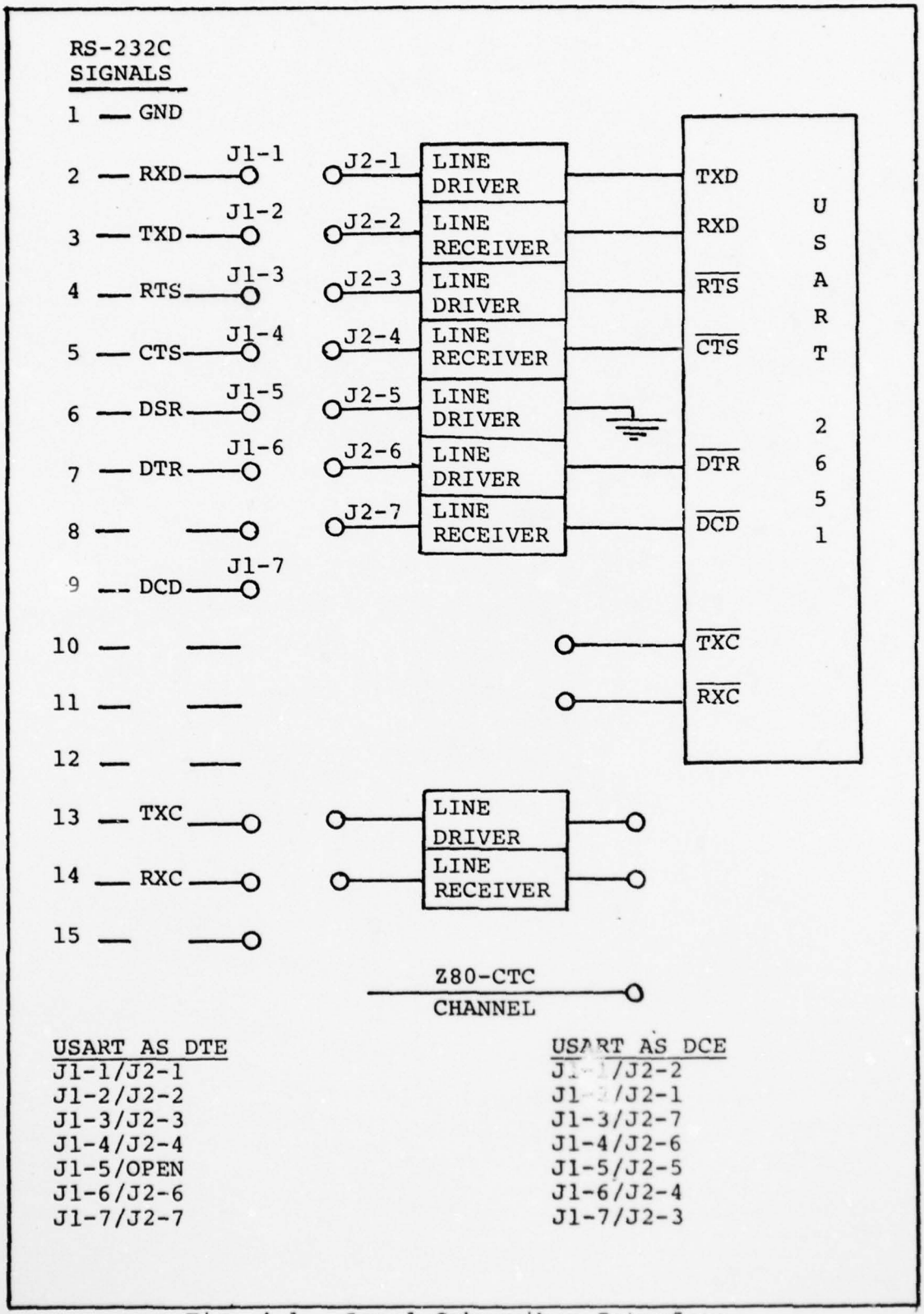
RS-232C Requirements. The RS-232C standard specified the signal characteristics between data terminal equipment and data communication equipment. To provide the universal network interface device with an RS-232C interface, the interface had to be classified with regard to these two categories. Applications were postulated which required the interface to satisfy both categories. As an example, it was conceived the universal network interface device could be collocated with a number of peripherals in which case the interface must function as a DCE. Correspondingly, there could be applications where the interface would be connected to the peripherals through modems and communication channels. In this latter case, the interface must function as a DTE. With regard to the RS-232C standard,

the universal network interface device had to be able to function as both a DTE and a DCE.

Functional Requirements. The most efficient solution to satisfy the other functional requirements was to utilize a USART. There was no reason to design special hardware to perform the recognition and parallelization functions when a low-cost device was readily available which would accomplish these functions. The typical USART performed start-of-message and end-of-message recognition functions for synchronous data, start-of-information and end-of-information recognition functions for asynchronous data and performed the serial-to-parallel conversion (Ref 23: 282-290). Many USARTs were available, most with very similar capabilities, thus making selection based upon technical criteria difficult. The USART finally selected was Signetic's 2651 Programmable Communications Interface. The 2651's functional capabilities included band rate generation, modem control and programmable operating modes. The USART supported BISYNC protocol with synchronous and delete character stripping and a transparent mode of operation. An asynchronous auto-echo mode may be programmed to accomplish reception and retransmission (echo back) of a received message without processor intervention (Ref 10: 65). These latter two characteristics were important in the selection of the 2651. This is not to say other USARTs do not have similar capabilities, but the ones evaluated for this investigation did not.

Peripheral I/O Port Design. Once the USART was determined, the design of the subscriber side of the USART was completed. The completed design is shown in Figure 4-1. The interface consisted of a 25-pin female connector, a row of jumpers to allow the interface to be configured as a DTE or DCE, line drivers and receivers to meet the RS-232C characteristics, and the 2651 USART. The secondary channel capability of the RS-232C specification was not developed for each port for the universal network interface device. To do this would have necessitated use of another USART at each I/O port dedicated to the secondary channel. Since the input board contained more than one I/O port and thus more than one USART, this secondary channel, if used, could be implemented using two I/O ports.

The design of the local subscriber interface was based upon the 2651's control signals. The control inputs which were significant were the $\overline{DCD}$ input which enabled the 2651's receiver and the $\overline{CTS}$ input which enabled the 2651's transmitter. For the case of the interface emulating a DTE, these inputs were identical to the complement of the RS-232 signals of the same name, thus all that was required was a line receiver to convert the DCE signal characteristics to DTE signal characteristics. The case of the interface emulating a DCE was more complex since the previous control signals must be outputs from the 2651. These control signals were developed from the $\overline{RTS}$ and $\overline{DTR}$ outputs of the 2651, applied to line drivers and connected by



USART AS DTE	USART AS DCE
J1-1/J2-1	J1-1/J2-2
J1-2/J2-2	J1-2/J2-1
J1-3/J2-3	J1-3/J2-7
J1-4/J2-4	J1-4/J2-6
J1-5/OPEN	J1-5/J2-5
J1-6/J2-6	J1-6/J2-4
J1-7/J2-7	J1-7/J2-3

Fig. 4-1. Local Subscriber Interface

jumpers to the CTS and DCD lines of the RS-232C connector. The $\overline{RTS}$ and $\overline{DTR}$ outputs of the 2651 are software controlled outputs which can be set or reset under software control. The RTS and DTR lines of the RS-232C signal connector were then used in conjunction with line receivers to enable the receiver ($\overline{DCD}$ input) and transmitter ($\overline{CTS}$ input) of the 2651 respectively.

Two other capabilities were incorporated into each port. Jumpers were provided to allow selection of the transmitter and receiver frequency source/sources for the 2651. The source/sources could be external to the 2651 through use of the RS-232C frequency lines. The frequency source/sources could also be provided by one channel of the on-board Z80A-CTC or by an internal source driven from the 5.0688 MHZ band rate input (BRCLK) to the 2651. These different frequency capabilities were provided in an attempt to meet the complete frequency range of operation required of the port. The other jumper capability allowed the selection of a 20 ma current loop input in lieu of the normal input.

Input Board Processor Interface. Now that the subscriber port was designed, the next step was to interface the port to the processor card. In this task, it was assumed the input card consisted of four I/O ports identical to the one in Figure 4-1. Given these ports, certain functions had to be accomplished to interface into the processor card. They were:

--The signals to and from the processor had to be buffered to maximize the number of cards which may use these signals.

--The processor had to address the individual registers of the 2651 and had to address the different 2651's.

--If an interrupt entry scheme was used, the processor needed to be provided the address of the 2651 service routine.

--If more than one interrupt occurred simultaneously, a device was needed to prioritize the interrupts.

The first part of the task involved the determination of the additional devices required to complete the functional design of the input card. The use of four 2651s had already been assumed. Each 2651 required an external frequency source derived from a channel of a Z80A-CTC. The Z80A-CTC was selected to achieve a measure of standardization of components between the processor card and the input card. This requirement established the need for two Z80A-CTC per input card. The only other device required other than buffering devices and an address decoder was a device to handle the interrupt entry technique.

Z80A Processor Interrupt Modes. The Z80A processor had three diferent modes of interrupt operation which are selected by execution of one of three interrupt instructions. In the maskable interrupt mode 0, the interrupting device is allowed to place one eight-byte instruction on the data bus for execution by the Z80A-CPU. The byte is normally a restart instruction which is an efficient one-byte call to any of eight subroutines located in the first 64 bytes of memory. In the maskable interrupt mode 1, the CPU does an automatic call to location 0038H and begins

executing the interrupt service routine at that point. In the maskable interrupt mode 2, the Z80A-CPU supports an interrupt vectoring instruction that allows the interrupting device to identify the starting location of the interrupt service routine. Mode 2 is the most powerful of the three maskable interrupt modes allowing an indirect call to any memory location by a single 8-bit vector supplied from the interrupting device. In this mode, the interrupting device places the 8-bit vector on the data bus in response to an interrupt acknowledge control signal. This vector then becomes the least significant 8 bits of an indirect pointer while the I register in the Z80A provides the most significant 8 bits. This address in turn points to an address in a vector table which is the memory starting address of the interrupt routine. Interrupt processing can thus start at any arbitrary 16-bit address of memory (Ref 24:7-8).

This latter mode of operation was selected for the interrupt entry scheme for the universal network interface device. The selection was based upon the memory flexibility of the technique and the fact that it allowed unique identification of service routines for any number of I/O ports. This technique required that the interrupt handling device have the capability to provide eight (two per 2651) unique eight-bit addresses.

Z80A Interrupt Acknowledgement. The other characteristics of the interrupt handling device for the input card

were dictated by the Z80A support chips and the Z80A interrupt acknowledgement method. The acknowledgement method for an interrupt consists of a special Z80A instruction cycle. After an instruction has been executed, the next instruction is normally fetched from memory. This normal instruction fetch cycle is identified by the $\overline{M1}$ output (pin 27) going low followed by the $\overline{MREQ}$ output (pin 19) going low. This cycle is modified to acknowledge an interrupt. For this latter case, the $\overline{M1}$ pin goes low identical to a normal cycle; however, slightly delayed, the $\overline{IORQ}$ output (pin 20) goes low (Ref 20:7-11 to 7-21). These simultaneously lows on the $\overline{M1}$ output and the $\overline{IORQ}$ output signify to all external devices that an interrupt is being acknowledged. It is now up to the devices to determine which of them with an interrupt pending has the highest priority.

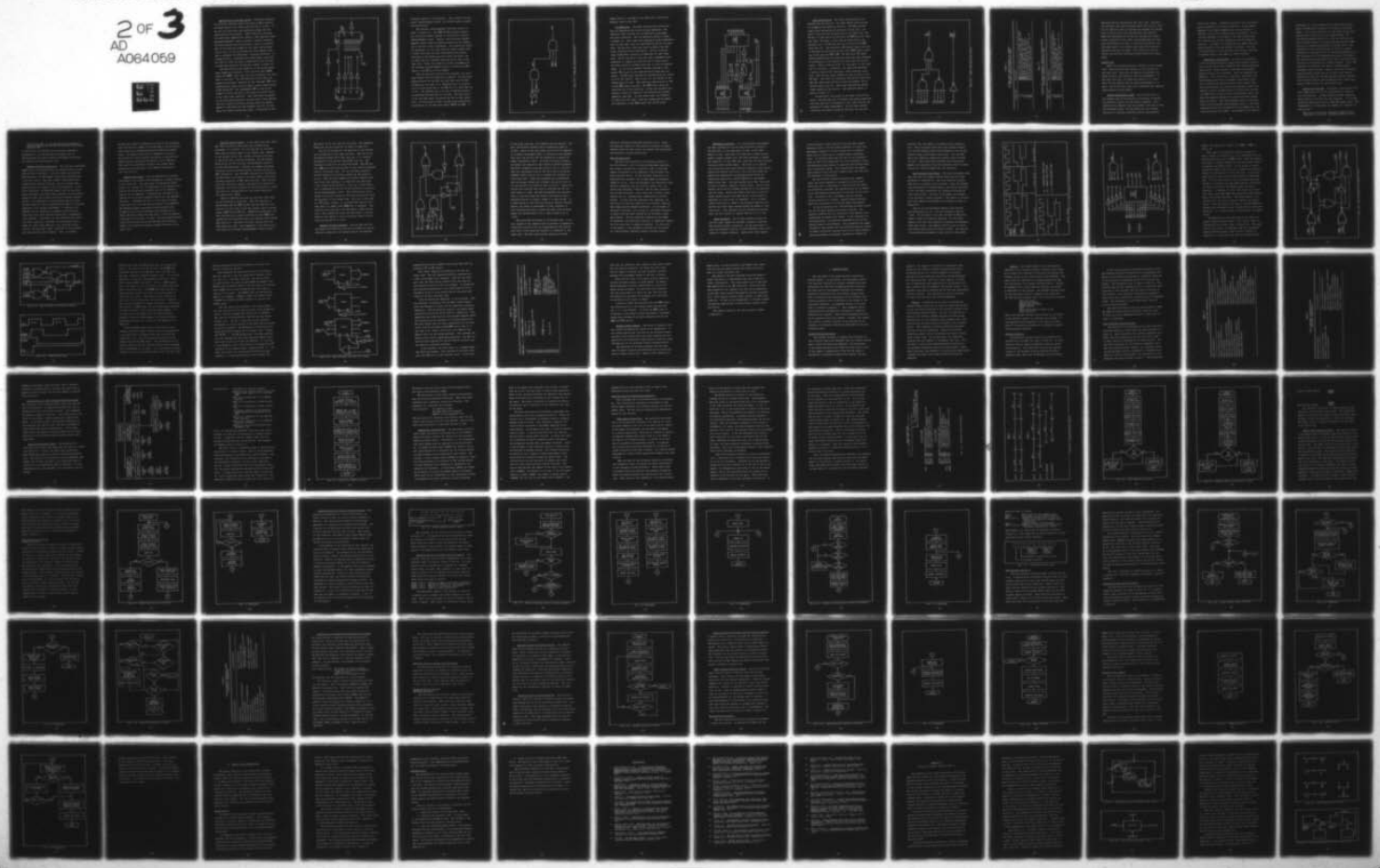
Z80 Daisy Chain. The prioritization technique supported directly by the Z80A processor and implemented through its support chips is the daisy chain technique. In this technique, priority is set by the location of the support chip in a daisy chain configuration. Each support chips' $\overline{INT}$ output is tied directly to the $\overline{INT}$ input of the processor. Each support chip has one additional input, Interrupt Enable In (IEI), and one additional output, Interrupt Enable Out (IEDO), which effects interrupt processing. To implement the daisy chain, the support chips's

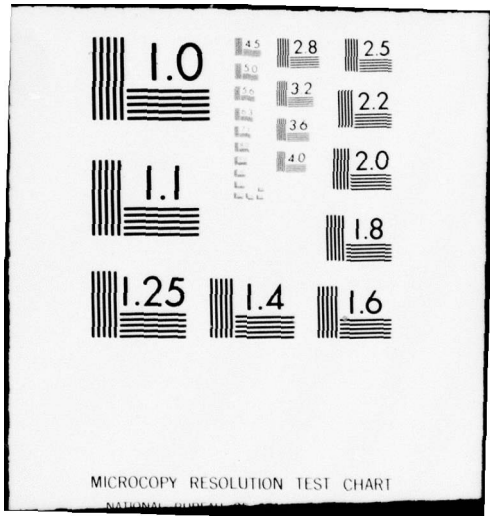
(with the highest priority interrupt) IEI input is tied to +5 volts to indicate it has the highest priority. The IEO output of the highest priority support chip is connected to the IEI input of the support chip with the second highest priority. This chaining of IEIs and IEOs continues until all support chips are included in the chain. Whenever a support chip in the chain generates an interrupt request, its IEO line goes low which in turn causes the IEOs and IEIs of all support chips further down in the chain to go low. When an interrupt acknowledgement occurs, any support chip with its IEI input low is disabled and cannot respond to the interrupt acknowledgement (Ref 24:8). This is an efficient technique for establishing priority provided the ripple time to change the IEIs and IEOs is not too long and provided there is a method to change the IEIs and IEOs after the interrupt has been serviced. In the Z80A processor case, the support chip whose interrupt is being serviced, determines by special hardware when the interrupt service routine has been completely executed. The special hardware detects the fetch from memory of a RETI (return from interrupt) instruction. Upon detection and the execution of this instruction, the hardware sets the IEO output high which reenables the interrupts of all support chips down the chain (Ref 24:17-22). If the priority controller on the input card is to take advantage of this system, it requires a daisy chain input/output and some method to determine the end of the service routine.

AD-A064 059 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
PRELIMINARY DESIGN OF A UNIVERSAL NETWORK INTERFACE DEVICE.(U)
DEC 78 S C SLUZEVICH

UNCLASSIFIED AFIT/GE/EE/78-41 NL

2 OF 3
AD
A064059




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART

M8214 Priority Interrupt Device. The device select for priority interrupt control was Intel's M8214 Priorit Interrupt Control Unit (PICU) (Ref 25:6-183 to 6-185). was the only PICU which was technically simple and thus operated with any processor. Other PICUs were available with additional capabilities; however, they required uni processor-dependent control signals to function properly The PICU selected had one deficiency. While it satisfie most of the functions needed for the input card priority control unit, it did not provide the 8-bit address neces sary to implement the Z80A mode 2 interrupt structure. accomplish this, a SN74LS412 multi-mode buffer latch (Re 22:7-502 to 7-506) was used in combination with the PICU A simplified version of the design is shown in Figure 4-

The 8214 PICU has the capability to prioritize amon eight competing interrupts. Thus, the four 2651s $\overline{\text{RXRDY}}$ output and $\overline{\text{TXRDY}}$ output could be used to generate the in rupts to the PICU. The PICU would prioritize among the competing interrupts, generate its own interrupt ($\overline{\text{INT}}$) simultaneously output the interrupt's unique identifica on $\overline{\text{A0}}$ through $\overline{\text{A2}}$. The complemented $\overline{\text{INT}}$ is used by the 74LS412 to latch the value of the eight bits which appe on its input lines (DI0-DI7). This value must correspo to the interrupting devices' low byte vector table addr This will require correlation between where the address appears in the table and the strapping used for the ca After the 74LS412 latches the input, it generates its

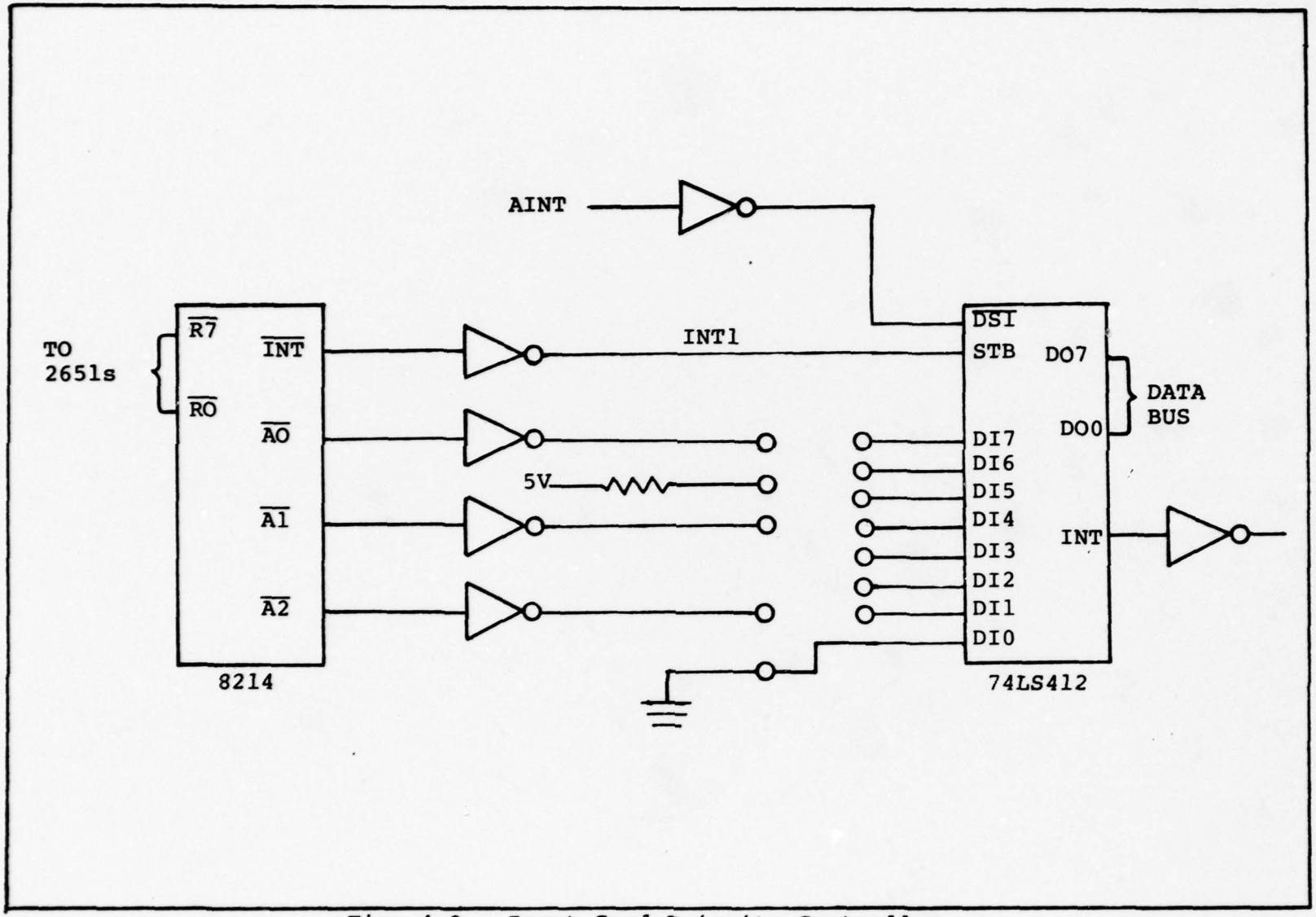


Fig. 4-2. Input Card Priority Controller

interrupt request to the processor. Upon receipt of a processor acknowledgement (AINT), the latched input is placed onto the data bus.

The interrupt acknowledgement detection circuit is shown in Figure 4-3. The $\overline{IORQ}$ and $\overline{M1}$ processor signals are inverted and ANDed together to develop the interrupt acknowledgement signal. However, this interrupt acknowledgement signal cannot be applied directly to the 74LS412 due to the daisy chain arrangement. Two conditional events are necessary: (1) the input card IEI (same as ETLG input to PICU) input must be high, establishing the pending interrupt as the highest priority within the chain, and (2) there must be a pending interrupt from a 2651 on the input card. These two conditional events are ANDed with the interrupt acknowledgement signal to develop the AINT signal to the $\overline{DS1}$ pin of the 74LS412.

Once the processor services the interrupt, the interrupt request from the PICU must be removed. The operating characteristics of the PICU are such that once the PICU processes an interrupt it is inhibited until it receives a low to high transition to its $\overline{ECS}$ (pin 23) input (Ref 20: 4-177). The simplest way to do this is through software by rewriting the mask word to the PICU. The other way is to use the AINT signal to provide the low to high transition. If this pulse is used, the mask word selected must be hardwired to the mask word inputs ($\overline{B0}$-$\overline{B2}$ and $\overline{SGS}$). A

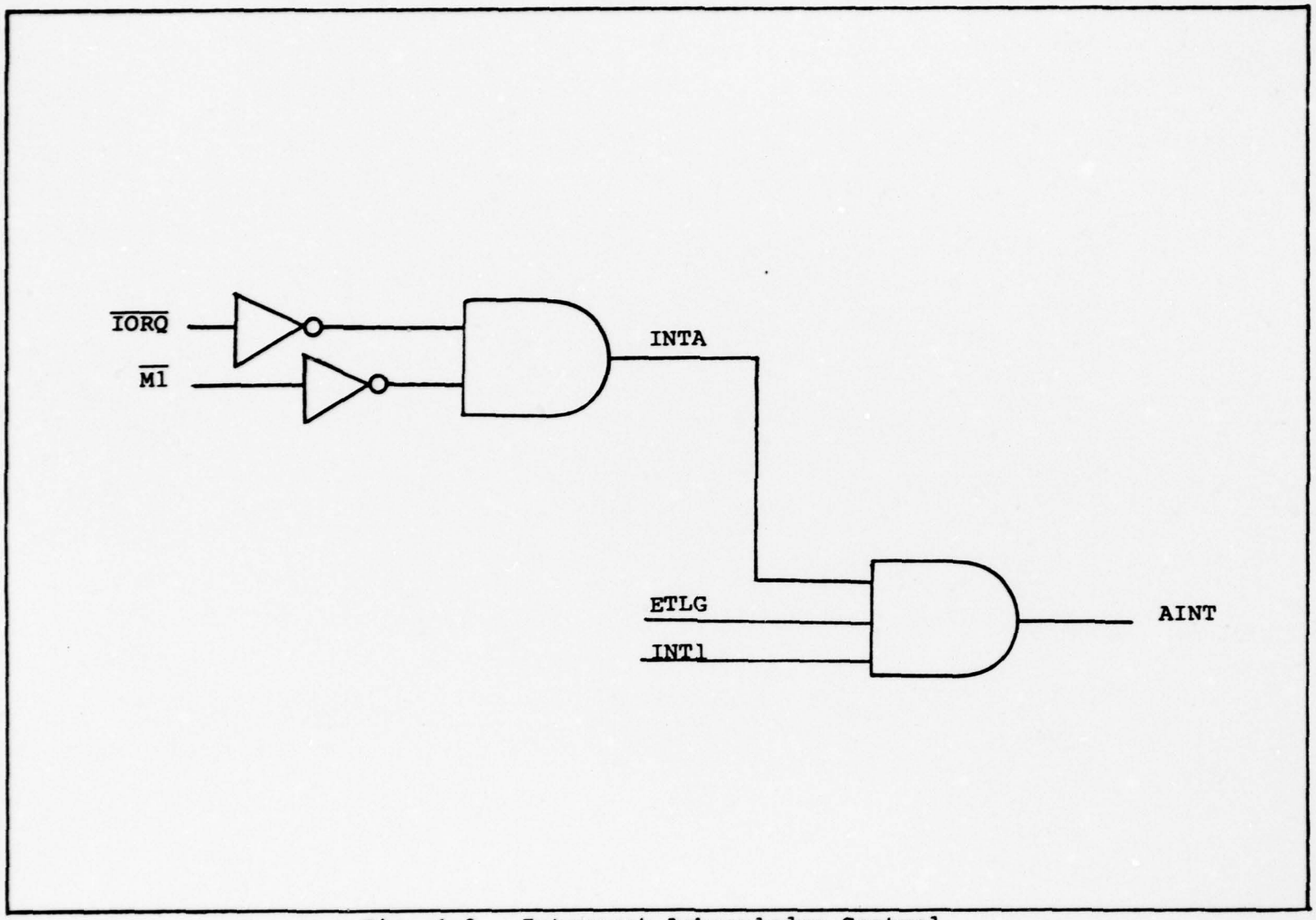


Fig. 4-3. Interrupt Acknowledge Control

jumper option is included on the input card to allow this hardware reset of the PICU.

I/O Addressing. The input card functions associated with the peripheral ports have now been completed. The remaining tasks involved the I/O addressing requirement and signal buffering. The I/O addressing design is shown in Figure 4-4. Each input card required 25 unique addresses--four for each 2651, four for each Z80A-CTC and one for each PICU. The A0 and A1 lines were used to address the individual registers within each device. The A2, A3 and A4 lines established a block of 32 addresses for each input card. The A5, A6 and A7 lines plus their complements were terminated at jumpers to allow the user to select where the block should be located within the 0 to 255 I/O address range. A 3 to 8-line decoder (74LS138) provided a chip enable ($\overline{CE}$) signal to the selected device based upon its inputs. The $\overline{CE}$ signal provided was determined by the inputs to G1 of the 74LS138. When G1 was low, the outputs of the 74LS138 were all high. This condition should exist as long as the $\overline{IORQ}$ output was high. The $\overline{IORQ}$ signal going low signified one of two conditions. Either the processor was issuing a valid I/O request or the processor was acknowledging an interrupt. In this latter case, the 74LS138's G1 input had to be high. This was accomplished by ANDing the complement of the $\overline{IORQ}$ signal with the $\overline{MI}$ signal.

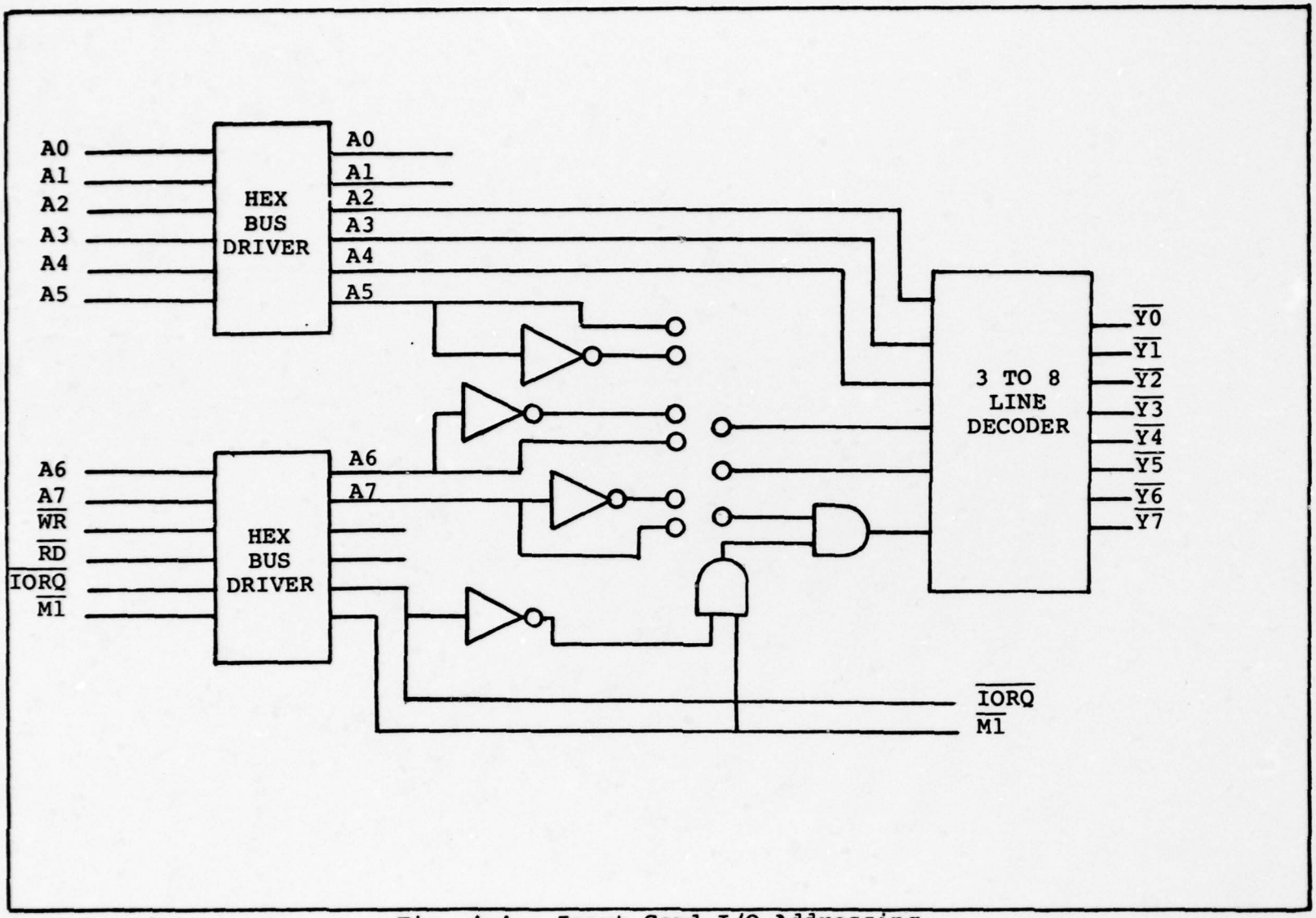


Fig. 4-4. Input Card I/O Addressing

Data Bus Buffering. The other bus which had to be buffered was the data bus. Two Intel M8216s were selected to accomplish the buffering function. The M8216 had two inputs which determined the direction of data flow. For data to flow from the processor bus to the input card, the $\overline{CS}$ (pin 1) input had to go low while the $\overline{DIEN}$ (pin 15) input went high. For data to flow from the input card to the processor bus, the $\overline{CS}$ input had to go low while the $\overline{DIEN}$ input went low. The processor's $\overline{WR}$ control signal was used to determine the direction of data flow ($\overline{DIEN}$ input) according to the equation $\overline{DIEN} = \overline{WR}$. The data flow ($\overline{CS}$ input) enable depended upon the different situations which required an interchange of data between the input card and the processor. These situations were: (1) interrupt, (2) I/O read and (3) I/O write. Internally, the input card developed a control signal which signified a valid interrupt situation. This control signal (AINT) was used for part of the $\overline{CS}$ input. The other two situations involved I/O operations. When the input card received a valid I/O request, one of the $\overline{CE}$ outputs went low. These then provided the second signal needed for bus control. The completed design is shown in Figure 4-5.

At this point, the design of the input card was complete. The next step was to evaluate the delays associated with the input card to determine if the input card met the processor's timing requirements. The two data transfer situations are shown in Table VI and Table VII with the

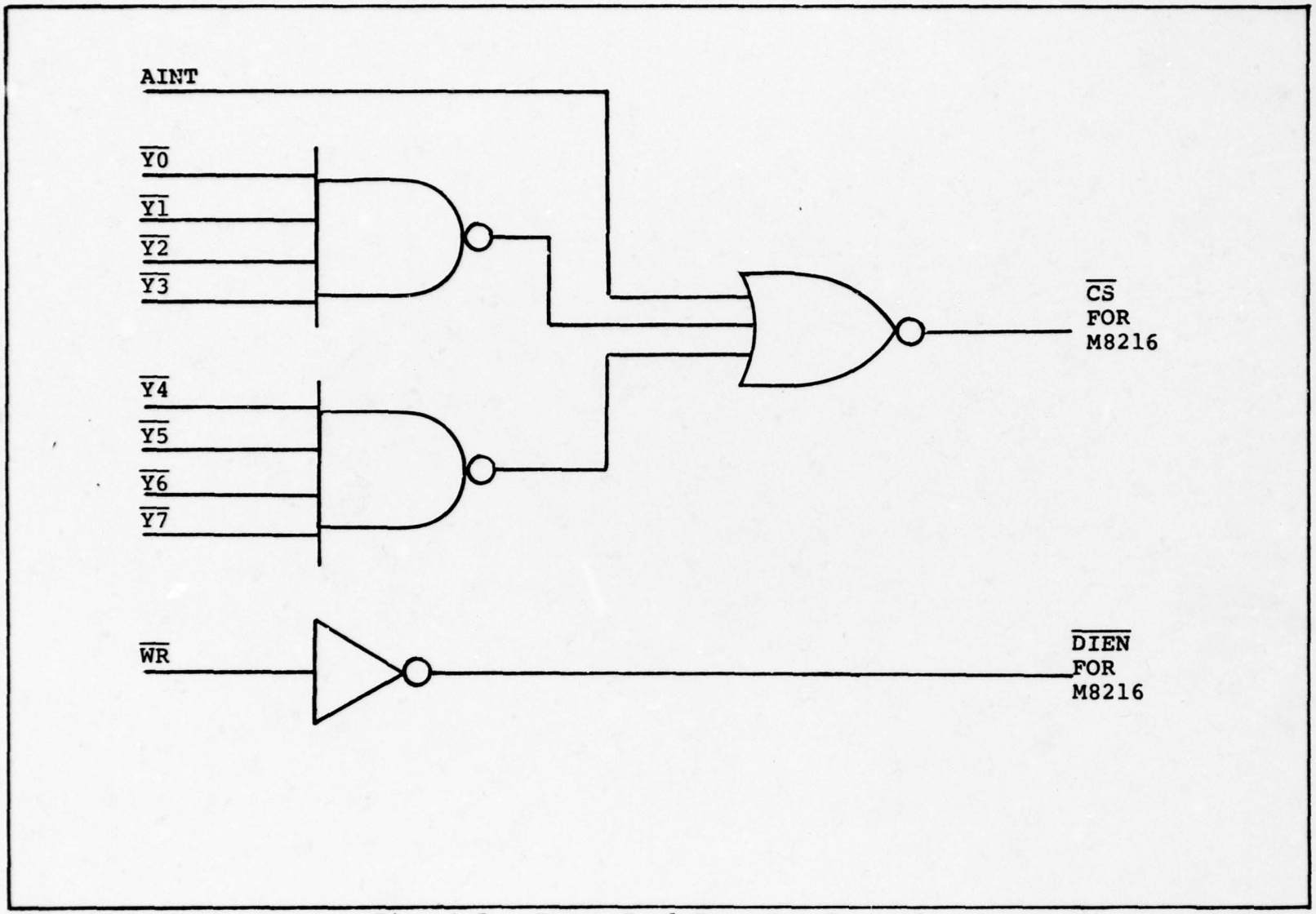


Fig. 4-5. Input Card Data Bus Control

TABLE VI

MAXIMUM DELAY FROM PROCESSOR I/O REQUEST
TO DATA AVAILABLE FROM INPUT CARD

Time (ns)	Occurrence
0	start of clock cycle 2 of a processor read cycle
75	$\overline{\text{IORQ}}$ delay from rising edge of clock (Ref 27:9)
97	$\overline{\text{IORQ}}$ delay through 74LS367A (Ref 26:6-37)
112	$\overline{\text{IORQ}}$ delay through NOT gate (Ref 26:6-2)
150	$\overline{\text{IORQ}}$ delay through AND gate (Ref 26:6-10)
188	$\overline{\text{CE}}$ delay through 74LS138 (Ref 26:7-136)
225	$\overline{\text{CE}}$ delay through the data bus control (Ref 26)
300	M8216 enable delay (Ref 25:6-192)

TABLE VII

MAXIMUM DELAY FROM PROCESSOR INTERRUPT ACKNOWLEDGEMENT
TO ADDRESS AVAILABLE FROM INPUT CARD

Time (ns)	Occurrence
0	start of first processor generated wait cycle
201	$\overline{\text{IORQ}}$ delay from falling edge of clock (Ref 26:9)
232	$\overline{\text{IORQ}}$ delay through 74LS367A (Ref 26:6-37)
302	delay to generate AINT (Ref 26)
317	$\overline{\text{CE}}$ delay through the data bus control (Ref 26)
392	M8216 enable delay (Ref 25:6-192)

associated delays introduced by the input card. The basic I/O read cycle and basic interrupt cycle (Ref 20:7-17, 7-19) for the Z80A were used to calculate the minimum time between start of the T2 cycle and time when data was received. These were determined to be 580 usec and 440 usec respectively. Although in both cases, the maximum delay was less than the minimum cycle time, in the latter situation, they were about equal if the delays associated with the microprocessor card were included. This could be an area of concern and should be evaluated further during the testing phase.

Network Card

Table III allocated different functions to the network card. These functions were similar to the functions of the input card. There was one important difference--the speed at which these functions must be accomplished. In the input card case, the speed was limited to 19.2 kb/s while for the network card the speed was bounded at 1.5 mb/s. One of the first tasks was to determine what approach would maximize network speed.

Network Transmission Speed. The limiting factor in the network case was the ability of the server to satisfy the different network I/O port service requests. This limitation was imposed since the processor was used to store the individual words. For the Z80A case, this storage utilizing an interrupt technique required approximately

20 usec per request. Assuming 8 bits/word, this translated into a transmission bit rate of 400 kb/s. If the network employed a half duplex communication link operating at 400 kb/s, the service requests could be satisfied. If the link was upgraded to full duplex, the Z80A could provide full service only if the transmission rate was reduced to 200 kb/s. If another full duplex link was added, the maximum transmission rate was reduced to 100 kb/s. The use of the processor to store the word thus caused a reduction in network speed as the number of links increased.

Word Storage Through DMA. To relieve the processor of the word storage function required the network card contain a device which would accomplish this without processor intervention. These devices, called DMA (Direct Memory Access) devices, were available. Their use presented two problems. First, there was a functional requirement that the processor perform the arithmetic calculations necessary to develop the network protocol error control word. Since the processor was storing the individual words, this function could be accomplished as the words were received. If a DMA device stored the word, the processor had to wait until the complete message was received and then perform this calculation. This after-message-receipt calculation would slow the message acknowledgement process. The other consideration was the incorporation of the DMA device into a multiprocessor environment. To accomplish this required

development of a bus controller which arbitrated all processor and DMA device requests for access to the shared memory. It was felt that this bus controller further complicated what started out to be a simple design. An alternative approach would be to limit the universal network interface device to a single processor and use a DMA device for reception/transmission of network data. This represented a very viable alternative since it allowed attainment of the upper bound for network transmission rate. The counterpoint to this approach was network throughput. The functions being performed by the other processors would now have to be performed by a single processor. Since this processor was required to do more, it seemed message throughput would decrease. However, there may be instances where throughput became less of a concern than network transmission speed and the DMA approach would be required. This suggested another card be designed which would incorporate a DMA function into the network card.

Network Card With DMA. The design of the network card with a DMA device will not be accomplished as a part of this investigation. It does represent another capability which should be available for user selection. It is recommended the DMA device used be a Z80A DMA support chip. This recommendation is based upon the comments in reference 20 which states:

> This is one of the most remarkable support devices described in this book. Although designed to work

with the Z80 CPU, it can--and should--be considered in any microprocessor system that transfers data blocks (Ref 20:7-78).

If this device is selected, the only design required to incorporate it into the network card would be to bi-directionalize the control signals and address bus between the network card and processor card.

Network Card Device Selection. Once the basic functional capabilities were estabished for the network card, the next step was device selection. Since the design utilized the Z80A processor, the Z80 support chips were evaluated first. It was felt that if there were support chips which would accomplish the network functions, the use of these chips would minimize the number of total chips required. In the final evaluation, the Z80A-SIO support chip not only minimized the number of external chips required, but also represented the best choice among the different network protocol devices. The Z80A-SIO was selected based upon the incorporated capability which allowed different modes of interrupt generation to be software programmed. In addition, the Z80A-SIO had the capability to generate eight different interrupt vector table addresses based upon a programmable vector address. Internally, the SIO identified the condition which required an interrupt to be generated, determined the vector table address of the interrupt service routine, and generated a mode 2 interrupt to the processor with this service routine address. Thus, no external

devices were needed to determine the cause of the interrupt. Since the SIO was a member of the Z80 family, the prioritization function was accomplished without additional support chips. The Z80A-SIO also had the capability to utilize the processor's block transfer instruction to provide half duplex message transmission/reception at up to 880 kb/s. This could be accomplished through a wait output which synchronized the processor to the Z80A-SIO transmission rate (Ref 28:1-27).

Network Card Design. Once the Z80A-SIO was selected, the design of the network card proceeded in a fashion similar to the input card. The input card I/O addressing design, Figure 4-4, was used for I/O addressing of the network card. Only two CE signals were required--one for the Z80A-SIO and one for the Z80A-CTC. A Z80A-CTC was included on the network card to provide frequency sources for the Z80A-SIO's two parts. A SN74LS90 (Ref 26:7-72) was used to provide clock inputs into the zero and one channel of the Z80A-CTC. The SN74LS90 outputs consisted of the processor clock divided by five and the processor clock divided by two. These additional inputs were provided to allow the Z80A-SIO to operate at rates above 175 kb/s. The Intel M8216 bidirectional data bus was used to provide data bus buffering; however, the controlling signals changed from that of the input card.

Data Bus Control Design. In the input card case, there were three situations where data bus information was exchanged: (1) interrupt, (2) I/O read and (3) I/O write. These same situations apply to the network card. However, in the network card's case, another situation arose due to the daisy chain interrupt technique. The Z80 support chips do not use the interrupt acknowledgement signal to set the IEO output high. Instead, the support chip whose interrupt is being serviced monitors the data bus for an RETI instruction. Once the RETI instruction is identified, the IEO output is set high on the first memory fetch cycle following execution of RETI (Ref 24:21-22). Since the network card contained a support chip which will generate an interrupt, the need to monitor the data bus during an interrupt had to be incorporated into the control portion of the network card's M8216.

When the Z80A-SIO generated an interrupt, there must be two directions of data flow. When the interrupt is acknowledged ($\overline{M1}$ and $\overline{IORQ}$ low/IEI high and IEO low), the direction of data flow is from the network card to the processor ($\overline{DIEN}$ low and $\overline{CE}$ low). Immediately afterwards, the data flow during any memory read must be from ($\overline{DIEN}$ high and $\overline{CE}$ low) the processor card to the network card to allow RETI detection. This data flow must continue until the IEO output goes high. This suggested a flip flop be utilized using the interrupt acknowledgement signal and the

IEO signal to set and reset the flip flop. The completed design for the bus controller is shown in Figure 4-6.

For the I/O situations, the operation of the controller is identical to the input card case. Once one of the chip-enable outputs ($\overline{Y1}$ or $\overline{Y2}$) goes low, it is inverted and applied to the OR gate causing $\overline{CE}$ to go low. The direction of data flow is controlled by $\overline{WR}$. When $\overline{WR}$ goes low, $\overline{DIEN}$ goes high allowing data to flow from the processor card to the network card. For the interrupt situation, the SIO requests an interrupt causing the IEO output to go low. This low pulse removes the preset condition from the D flip flop allowing it to duplicate the input upon a low to high clock transition. This low to high transition is generated by the low to high transition at the end of the acknowlegement signal. The transition set the $\overline{Q}$ output to one thus setting the direction of data flow from the processor to the network card. This one is also used as a $\overline{CE}$ signal, however, it is conditioned upon the fact that there is a memory read ($\overline{MREQ}$ low) in progress. This situation continues until the SIO detects a RETI instruction. This instruction causes the IEO output to go high which in turn applies a low to the preset input of the D flip flop setting it to a one.

Network Interface Standard. Once the data bus control design was completed, the last consideration was the interface standard for the network side of the Z80A-SIO.

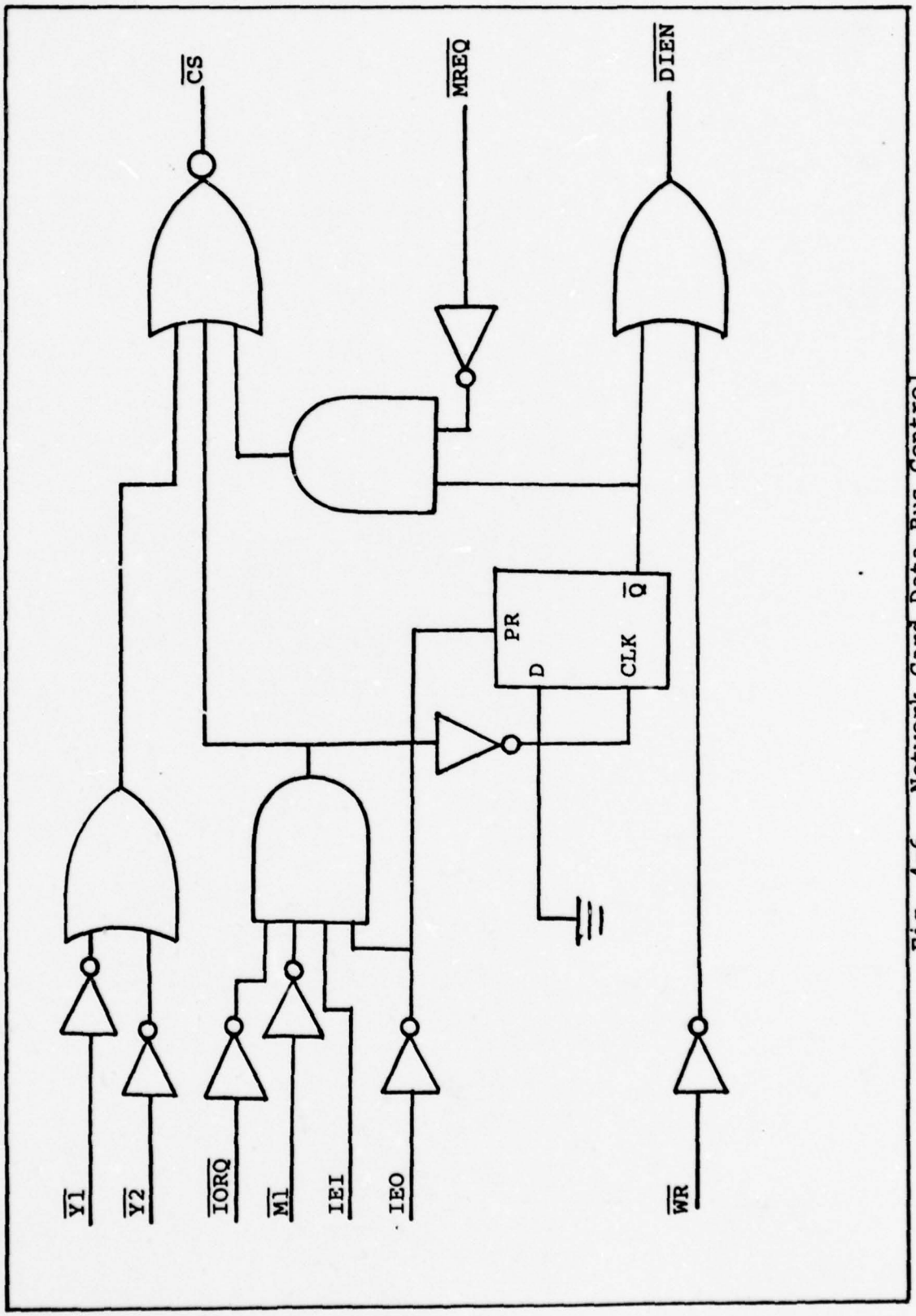


Fig. 4-6. Network Card Data Bus Control

In the input card case, the standard used was RS-232C. However, the RS-232C standard limits transmission speed to 20 kb/s. While it was conceivable the SIO will be employed in a network environment which is limited to this transmission rate, the SIO also has the capability to operate at higher transmission rates. To allow this later situation, the output was required to meet RS-422 or RS-423 standards. The RS-423 standard allows data rates of up to a 100 kiloband over unbalanced circuits (Ref 29:2) while the RS-422 standard allows rates of up to 10 mband over balanced circuits (Ref 30:3). These two standards were incorporated into the design through use of MC3487 line driver (Ref 31: 82) and the 9637 line receiver (Ref 32:11-217). These line drivers and receivers were used to configure the SIO network output as a DTE. The DTE configuration was selected since it was envisioned the network side would be transmitting/receiving to either a modem or a cable system. If a cable system is used with the universal network interface device, the reader is encouraged to study reference 33. This reference describes a tested interface which provides proper bit synchronization over a cable system at up to 1 mb/s.

This concluded the design of the network card. A circuit diagram of the complete card is provided in Appendix B. A detailed evaluation was not accomplished on the network card since it was basically identical in operation to the input card. The most time-critical operation occurred

during an interrupt acknowledge processor cycle. Since the important signals in this operation traverse an almost identical path for both cards, the network card should meet the processor's timing restrictions.

Dual Processor Card

The last card required for the hardware portion of the interface was a card to allow multiprocessor operation. The original concept was to develop a card to allow any number of processors to be employed in the universal network interface device. As the different functions to be performed were analyzed, the optimum number of processors seemed to be two. With two processors, the functional tasks could be segregated into two distinct groups--one, concerned with peripheral functions, and the other concerned with network functions. Since there was a distinct break between the two, interprocessor communications would be minimal. If more than two processors were employed, the allocation of fucntions would not be as distinct requiring more communications between processors. As the number of processors increased, the lock-out of individual processors as global data was being changed by one processor became more complex. The bus controller required as the number of processors increased would increase in complexity causing the life cycle cost to change accordingly. At this point in the design, it was decided to provide only the option of a two-processor universal network interface device.

Z80A Memory Reference. In a two-processor environment, the basic problem was to allow both processors access to the same data, at the same point in time, in the least amount of time. There had to be some way to delay one processor's memory request until the other processor's request was completed. In the Z80A case, there are two basic instruction cycles which effect memory. The first is an instruction fetch cycle (M1) which normally requires 4 clock cycles. During this machine fetch cycle, the first half reads the memory word addressed by the program counter while the second half generates a refresh address for any dynamic memory being used. The other machine cycle (M2), data read or write to memory, requires 3 clock cycles. Each of these machine cycles can be extended through use of the Z80A wait (pin 24) input. During the second clock cycle of the different machine cycles, the Z80A checks its wait input to determine if a wait state is requested. If so, an additional clock cycle is added to the executing machine cycle and the wait input again checked during the middle of this clock cycle. This checking and wait generation continues until the wait request is removed (Ref 20:7-15 to 7-16).

Basis of Design. The two basic machine cycles and the wait input capability provided a method to arbitrate dual processor memory references. At any given point in time, one processor could be in seven different states with regard to a memory reference. These seven states equate

to the different cycles involved in the two basic memory reference cycles. For the second processor to access this same memory requires it to be in cycle one of a machine fetch cycle or cycle one of a machine memory read/write cycle. It has been shown (Ref 34) that if one processor's clock is 180° out of phase with the other processor's clock the processors could operate in parallel with minimum reduction of processor speeds. This required that the memory being used be static and have a memory cycle time less than the processor's clock period.

For the universal network interface case, however, these two factors equated to increased life cycle cost. Since there was a requirement to locally store the different information received, the characteristics of the memory to be used impacted significantly on the cost of the interface. To minimize this, the memory used for message storage should be the slowest, cheapest memory available which was consistent with processor speed requirements.

To try to minimize cost, the different processor states for the two basic machine cycles were analyzed. The analysis revealed that the instruction fetch machine cycle established the memory cycle speed. In the universal network application, the functions performed by the two processors tended to be different. This suggested that common instruction code between the two processors would be minimal. If common instruction routines could not be shared between the two processors and dynamic refreshing of memory was

required, then the number of allowable states changed to five. These allowable states were then analyzed (Figure 4-7) and the minimum access time determined to be one and one-half times the processor clock cycle. For the Z80A case, this equated to memory with access times of around 370 usec. Since these were available utilizing dynamic memory, the stipulation that the processors could not share instruction routines seemed very cost effective.

Dual Processor Card Design. The design proceeded based upon the need for a memory refresh signal and the ideas presented in reference 34. The first decision involved what portion of memory would be shared. Since the Z80-MCB card provided 8K of on-board memory, this 8K was allocated to the individual processors for instruction storage and local data storage. The rest of the memory was assumed to be available to both processors. The option to allocate more local memory was provided in the design as shown in Figure 4-8.

Each of the two processors' address lines (A0-A15) were terminated at 2 to 1 line data selectors (Ref 26-7-181). For the A0-A11 line case, the NOT and AND gates shown in Figure 4-8 were not required. For the other address lines they were required to identify the addresses which were shared. The jumpers allow the user to select what address space above 8K could be shared. Any time one of the processors attempts to gain access to this shared

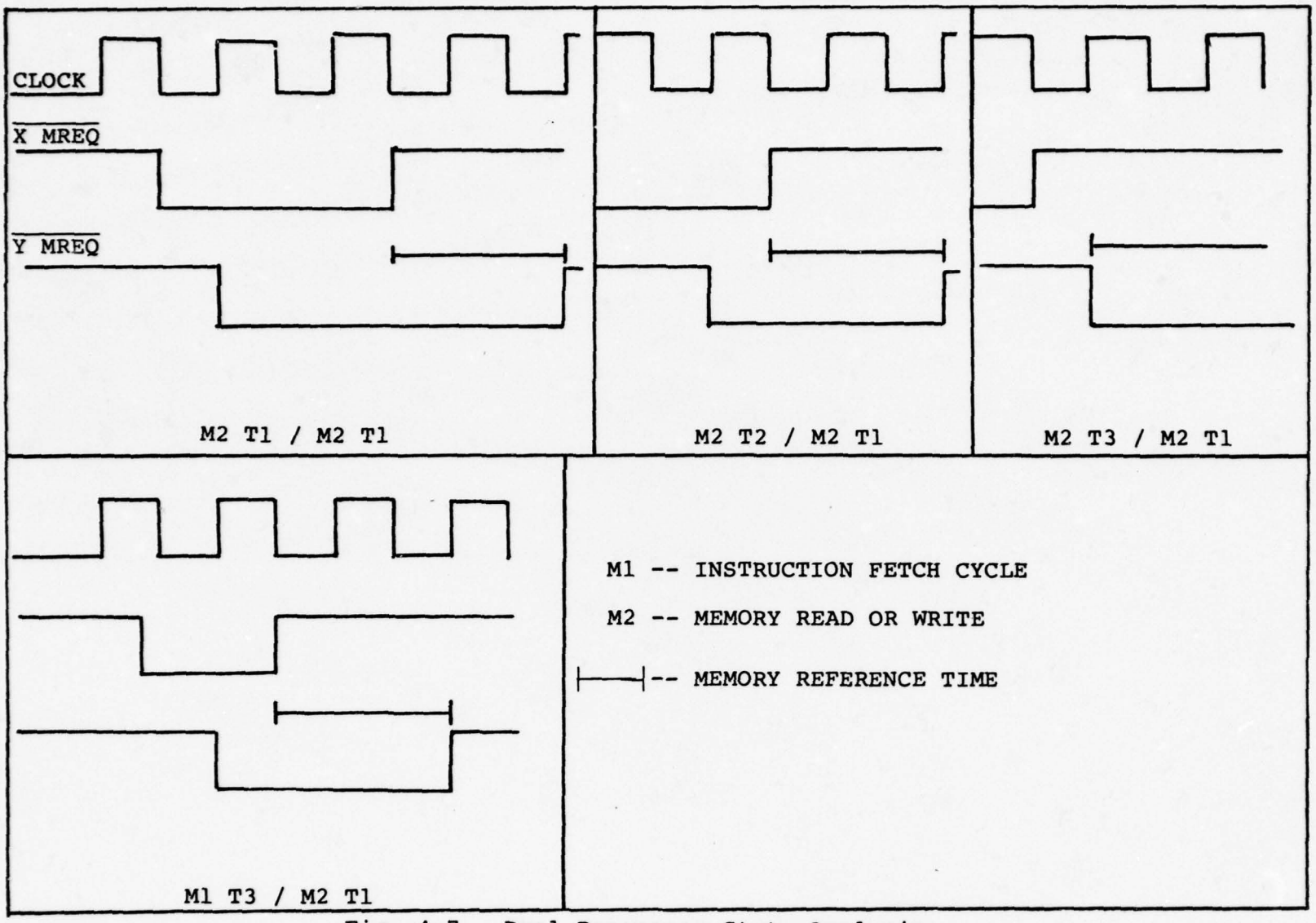


Fig. 4-7. Dual Processor State Analysis

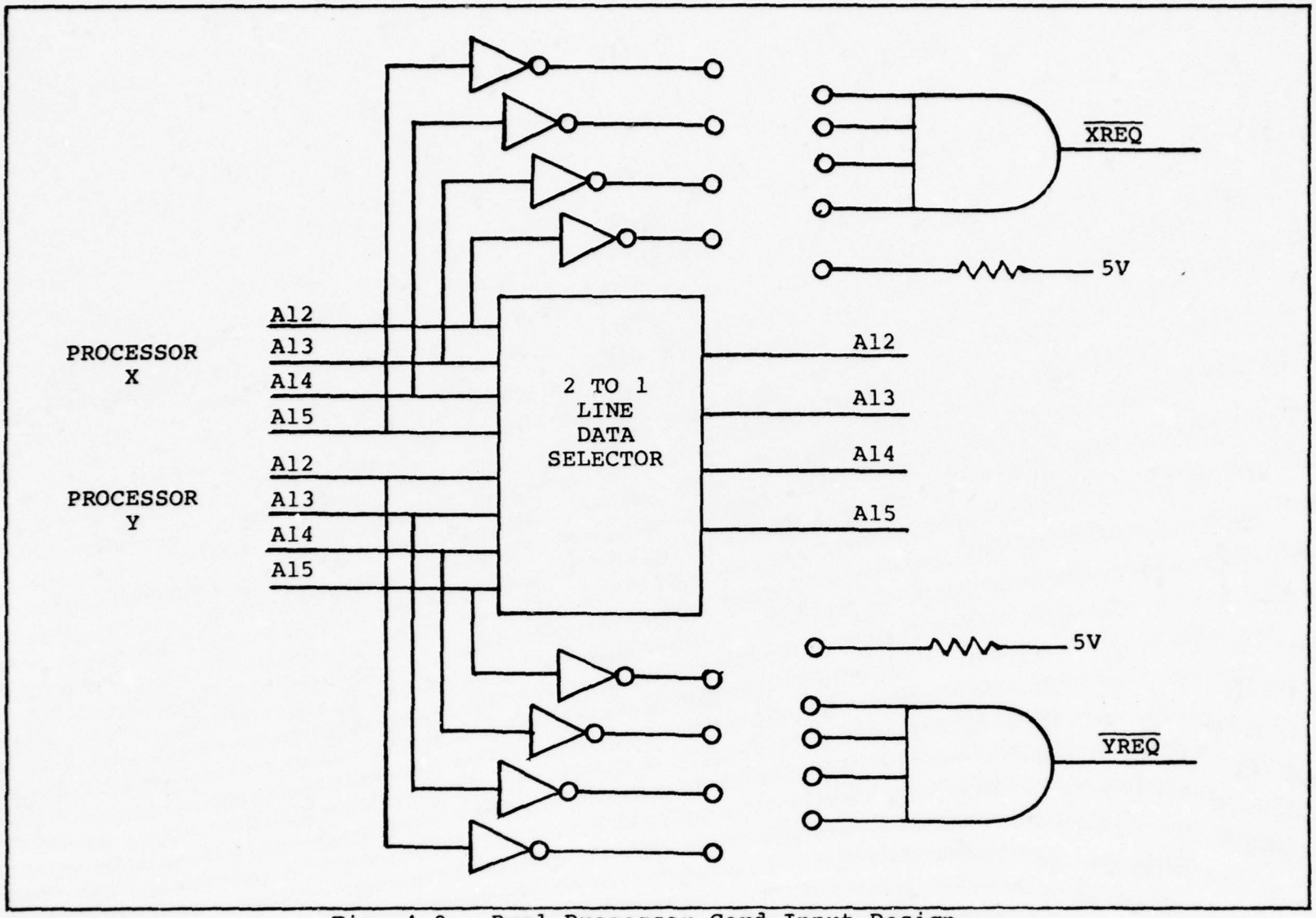


Fig. 4-8. Dual Processor Card Input Design

memory, the appropriate request line ($\overline{\text{XREQ}}$ or $\overline{\text{YREQ}}$) is driven low.

This request is processed as shown in Figure 4-9. When a request is generated, it is not processed until the processor generates the low $\overline{\text{MREQ}}$ signal. When this occurs, the request is passed through the tri-state buffer (74125) to generate the SELECT signal. As the signal is passed through the SN74125 (Ref 26:6-33) it biases the second processor's tri-state buffer to the off-state. It also provides one-true input to the second processor's WAIT NAND gate. If at a later point in time, the second processor attempts to reference the same shared memory, the SN74125 off state will prevent the SELECT signal from being generated. In addition, it will also provide the second true input to the WAIT NAND gate generating a WAIT request back to the second processor. This WAIT condition will continue until the first processor completes the memory action. When this occurs, one input to the WAIT NAND gate becomes false removing the wait request back to the second processor. It is then allowed to continue with its memory action.

The SN74125s are also controlled by the other processor's memory refresh signal. This control signal is developed by the circuit shown in Figure 4-10. The operation of this circuit is dependent upon the relationship between processor signals (Ref 22:8-10) as shown in Figure 4-11. During the first part of the memory fetch

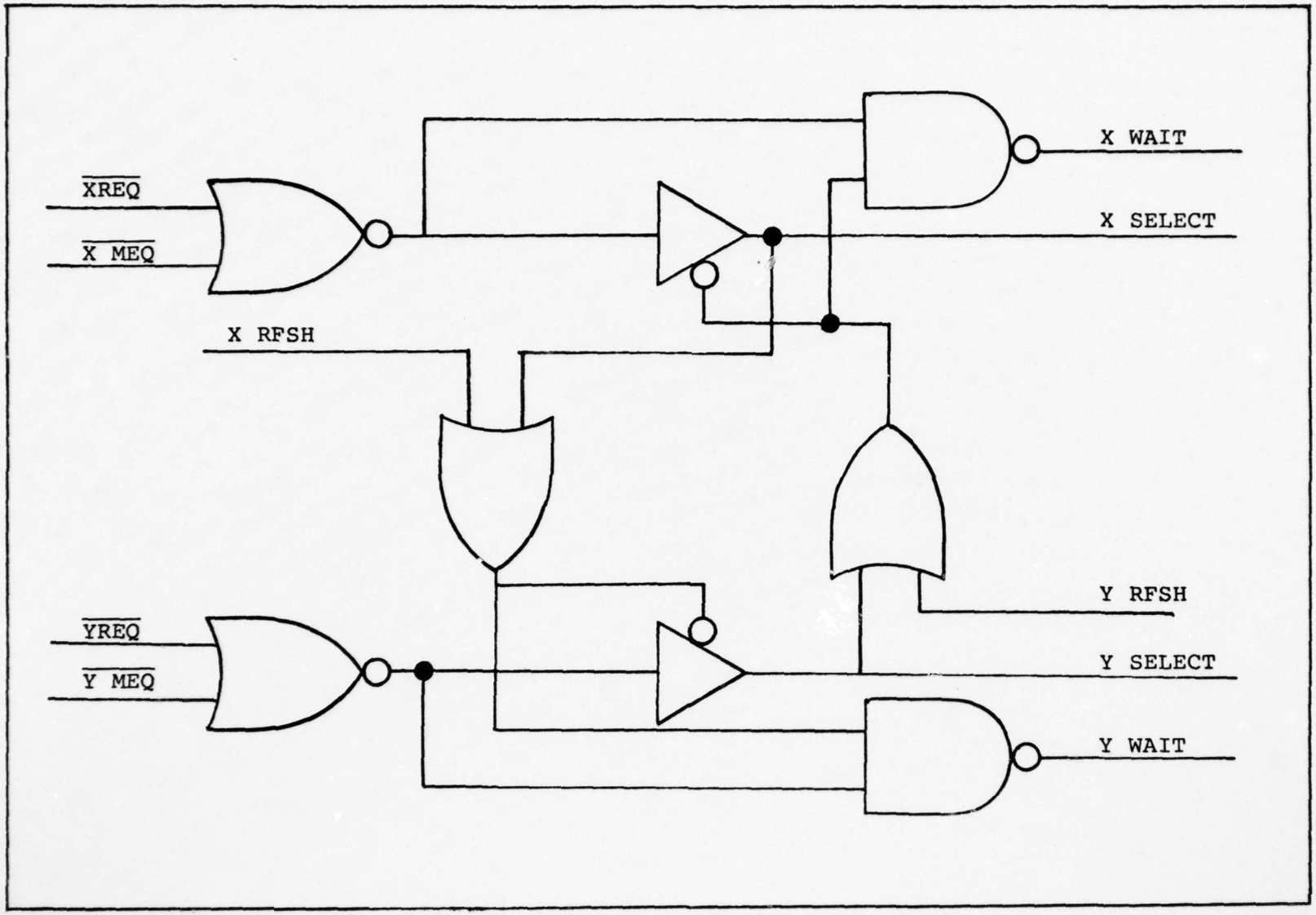


Fig. 4-9. Dual Processor Card Request Design

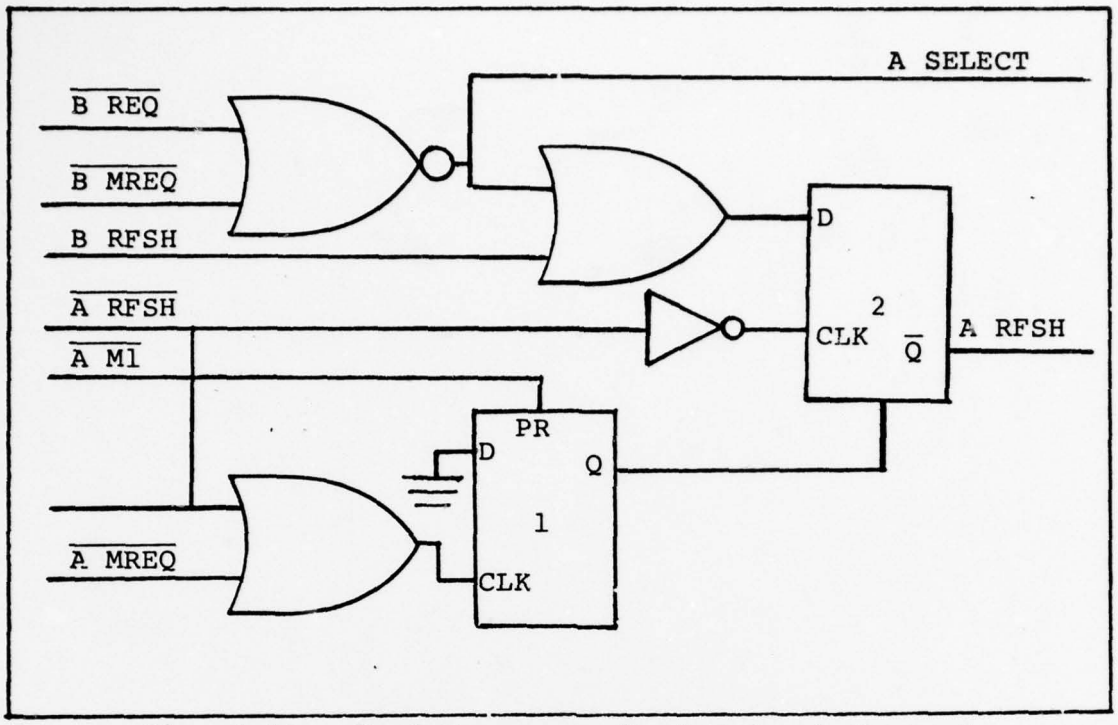


Fig. 4-10. Dual Processor Card Refresh Control

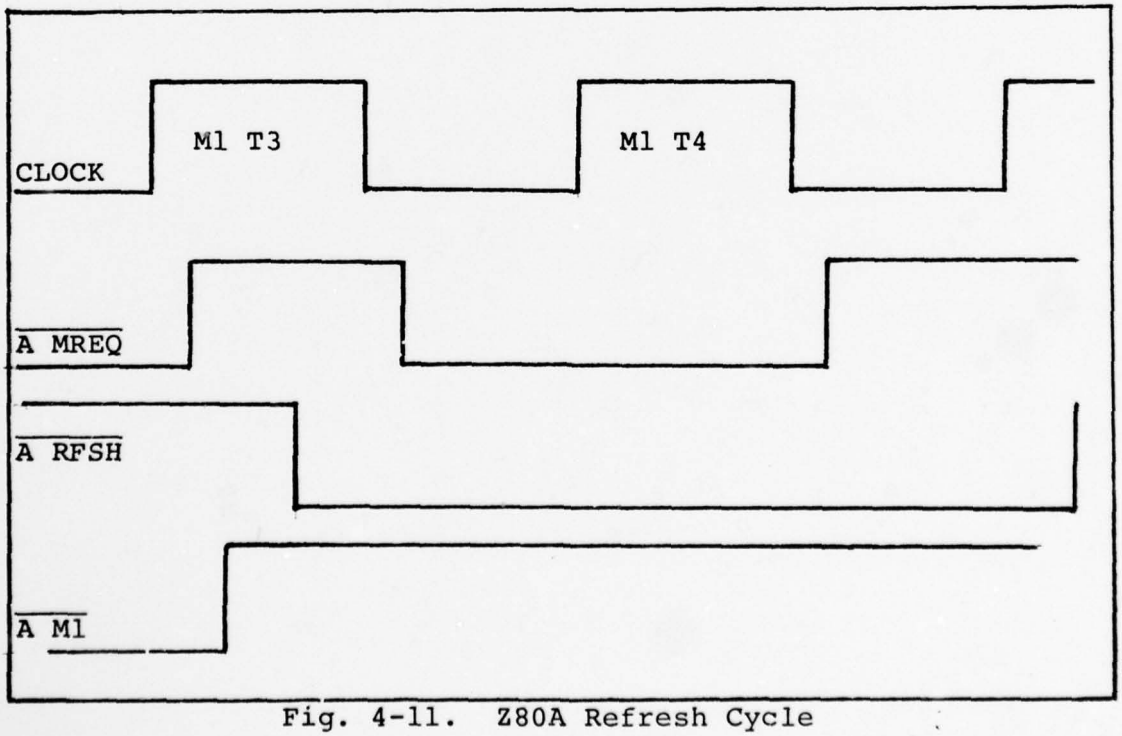


Fig. 4-11. Z80A Refresh Cycle

machine cycle (M1) the $\overline{\text{A M1}}$ signal goes low setting flip flop 1. At a point in the M1 T3 cycle, the $\overline{\text{A RFSH}}$ goes low which, because of the NOT gate, sets flip flop 2 to the other processor's memory reference state. If the other processor is using the shared memory, flip flop 2 is set which causes A RFSH to be false. If not, A RFSH becomes true. The states of the flip flop do not change until the end of the M1 T4 cycle. As $\overline{\text{A MREQ}}$ goes high, the low to high transition causes flip flop 1 to be reset which in turn sets flip flop 2. This condition will continue to exist until the next M1 cycle sets flip flop 1 allowing the next refresh signal to change flip flop 2. This circuit thus provides a $\overline{\text{RFSH}}$ signal for the shared memory provided the other processor is not using the shared memory. If at some point in the RFSH cycle, the other processor attempts to use the shared memory, the lock-out process described previously will occur until the RFSH cycle is completed.

Since the RFSH signal does not have priority over another processor's memory actions, there is a possibility a row would not be refreshed within the allocated time (typically 2 ms). This was minimized by having two processors provide the RFSH signal. This possibility can be further reduced since the refresh register within the Z80A can be programmed to any value. If one processor's refresh register is set to zero and the other to 64, the total time

between processor-generated refreshes for any given row would be reduced by one-half.

This completed the arbitrator portion of the design. The next step was to use the arbitrator's signals to control the 2 to 1 line data selectors. The data selector is controlled by two inputs called STROBE and SELECT. The SELECT input determines which of the two inputs are selected while STROBE (active low) determines when this input is applied to the output. The STROBE input was developed by NORing the X SELECT, Y SELECT, X RFSH, and Y RFSH signals. X SELECT and X RFSH were NORed together to generate the SELECT signal.

Once the address was provided to the shared memory, the next step was to route the data back to the proper processor. This was accomplished through use of SN74365A (Ref 26:6-36) and SN74367A (Ref 26:6-36) hex bus drivers (Figure 4-12). The use of these bus drivers dictated development of control signals to determine which processor the data was to/from. The SN74365A are controlled by two inputs $\overline{G1}$ and $\overline{G2}$ according to the formula input = output when $\overline{\overline{G1}\ \overline{G2}} = 1$. The $\overline{WR}$ and $\overline{RD}$ signal from each processor plus the complemented arbitrator-generated SELECT signals were used directly to control the SN74365A. In the case of the SN74367A, four of the drivers are controlled by $\overline{G1}$ according to the formula input = output when $\overline{G1} = 0$. The other two drivers are controlled by $\overline{G2}$ using the same condition. To develop the control signal needed, the

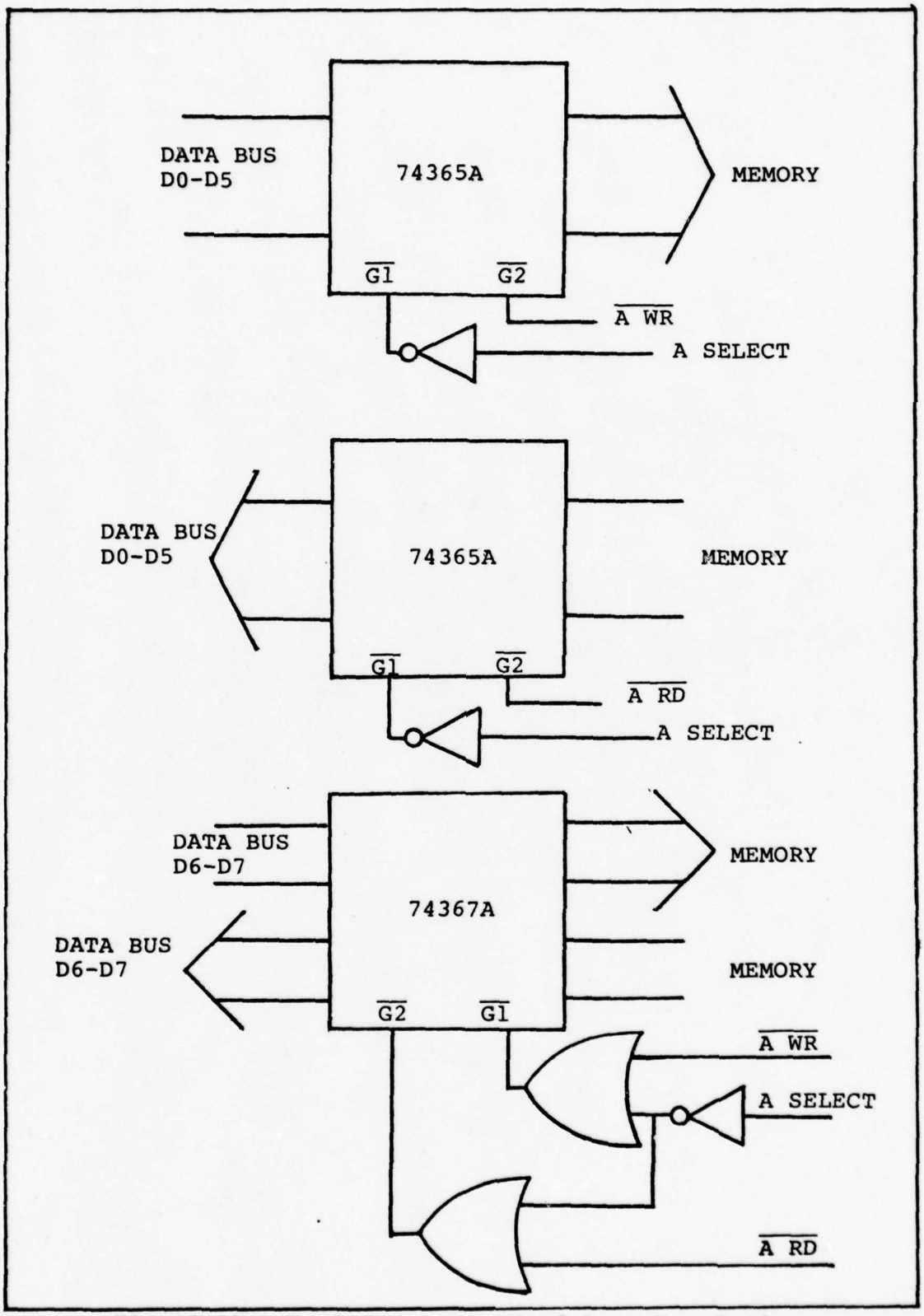


Fig. 4-12. Dual Processor Card Data Bus Design

complemented arbitrator SELECT signals were ORed with the processors $\overline{RD}$ and $\overline{WR}$ signals.

This almost completed the design for the dual processor card. One very important function, generation of a 180° out-of-phase clock remained. This was very important since the whole design rested on the fact that the two processor clocks are 180° out-of-phase. To accomplish this required modification to the clock input of one of the MCB to allow it to be driven from the inverted clock output of the dual processor card.

Once the design was completed, it was evaluated. This was necessary to determine how the memory speed requirements had changed as a result of the additional arbitrator components. From the previous analysis, the memory speed requirements were set by the M1 T3/M2 T1 combination. This combination was used along with the maximum component delay from reference 26 and reference 27 to develop Table VIII. The table showed that the delays associated with the arbitrator card and the Z80A minimum $\overline{WAIT}$ set-up time of 70 usec (Ref 27:9) caused a wait cycle to be added to the machine cycle for the second processor. This extended the memory access time to approximately 600 usec. The same calculations were repeated for the M2 T2/M2 T1 situation and the identical situations occurred.

In the analysis, the worst case; i.e., maximum component delay was assumed. This resulted in the best situation, the addition of a wait machine cycle. If this wait

TABLE VIII

DUAL PROCESSOR CARD EVALUATION
M1 T3/M2 T1

Time (ns)	Processor X Event	Processor Y Event
0	start of M1 T3	not referencing shared memory
125		start of M2 T1
130	$\overline{\text{XRFSH}}$ goes low	
162		Y SELECT disabled
171	refresh address available	
250	start of M1 T4	
283		$\overline{\text{Y MREQ}}$ goes low
298		$\overline{\text{Y WAIT}}$ goes low
368		wait valid at processor
375		start of M2 T2
405	$\overline{\text{X MREQ}}$ goes low	
453		$\overline{\text{Y WAIT}}$ goes high
455		Y SELECT goes high
484		address available
500	end of M1 T4	processor checks wait input
533		high wait valid
625		enter M2 wait
875		enter M2 T3
1000		input

cycle was not generated, then a memory access time of about 325 usec would be required. To insure this wait cycle could be added if desired, the dual processor card was designed with strappable delays in the wait circuitry.

Given the fact that a wait cycle could be added for any dual processor access to shared memory, the memory speed requirement would be established by a single processor access to memory. For the worst case situation, this was calculated to be about 375 usec from the time the address was available at the memory until data must be available on the output of memory.

A review of Table VIII revealed that the $\overline{\text{MREQ}}$ signal was generated prior to the address being selected from the 2 to 1 line decoders. To allow the $\overline{\text{MREQ}}$ signal to be delayed to meet memory timing requirements, strappable delays were included on the dual processor card for the $\overline{\text{MREQ}}$ and $\overline{\text{RFSH}}$ signals.

Hardware Design Summary. The design of separate cards now allowed the modularity concept to be implemented. In any network application, the user of the universal network interface device could select the cards necessary for his application and interconnect these cards to form his unique configuration of the universal network interface device.

The operation of the dual processor card was such that the memory required was determined by the single processor's memory access time. Should a dual reference to

memory occur, the dual processor card added a wait cycle thus insuring the memory access time would not be less than in a single processor case.

The design of the dual processor card also seems to permit inclusion of a DMA network card into a dual processor configuration. This would require use of the Z80A-DMA support chip as the DMA controller. Since this chip does respond to wait requests, the chip can be controlled by the same output signals developed by the dual processor card. This requires further study; however, it seems very promising. If this can be accomplished, it further extends the possible applications of the universal network interface device.

The complete design of the entire system is shown in Appendix B.

V. Software Design

The last phase of the design process involved the software design. In this phase, the requirements definition functions selected for software implementation were translated into code which accomplished those functions. The first part of this chapter discusses the different constraints associated with the software design effort. This is followed by the segregation of the software functions by processors and a discussion of the design of the individual functions. An assembled version of the software is provided in Appendix C. This assembled version contains detailed documentation necessary to completely understand the software. This detailed documentation will not be repeated within this chapter. Instead, this chapter will provide a general overview of the structure of the software, the different subroutines developed and the data structures used.

Software Design Constraint

The software necessary to operate the universal network interface device was dependent upon the network environment in which the device was employed. The particulars of the network protocol used along with other factors such as the number of communication links and the types of peripherals interfaced influenced the software that was

required. The number of variations in peripheral types along with the different network protocols which could be encountered did not allow development of universal routines for those functions which were network-dependent. For those functions, there was a need, however, for software to demonstrate the capabilities of the universal network interface device's hardware and software design. This software could then be modified by the user and incorporated into his programs. This approach was used in the design of those functions which were network-dependent.

Testing. An important factor which influenced the software design effort was the need for simplified software to test the proper operation of the universal network interface hardware/software design. To test the complete features of the universal network interface hardware design dictated that all the different cards (network card, input card and dual processor card) be included in the software effort. This required an operating system be developed for each of the two processors. Within the different operating systems, certain techniques were used to accomplish a given task. For the most part, the techniques selected were the simplest to accomplish that task. This was done to allow easier hardware/software isolation of any problems encountered during the testing phase. While these techniques were adequate for testing, user application programs may require more sophisticated techniques be employed.

Protocol. One simple method to test the complete operation of the universal network interface device would be to connect two terminals to the device and connect the transmit output to the receive input of the Z80A-SIO. This would allow the terminals to exchange messages and thus test the design of the network interface device. However, to implement this testing approach required a structure be developed for the message. The message structure used for the operating systems in Appendix C was based upon the SDLC message structure (Ref 35:1-1) and was as follows:

Flag (01111110)
Destination Address
Message Identification
Sender Address
Text
Error Check--CRC-16 Preset to One
Flag (01111110)

Where this message structure had an affect on the design of the operating system, the software was so noted. If the suggested testing approach is not used, then those parts of the operating systems can be changed to support the new message structure.

Software Functions

The different functions which were selected to be accomplished in software are shown in Table IV. To these functions must be added an additional function, device initialization. This additional function was required as most of the hardware chips selected had different operating modes which were established through software.

If the testing approach previously discussed is used, the different software functions had to be segregated into those to be performed by processor #1 (operating system #1) or the processor #2 (operating system #2) or by both processors. The segregation used is shown in Table IX. This criteria used for this segregation was to isolate the input functions of the interface from the network functions of the interface. This minimized interprocessor communications since each processor was performing mostly independent tasks. This segregation also allowed easier isolation of any software problems.

The table also demonstrates the effectiveness of SADT. The SADT has modularized the different functions, each of which can now be implemented through a short block of code or a subroutine.

Input Processor Operating System

The functions to be performed by the input processor's operating system are listed under the input processor in Table IX. These functions can be further segregated into those functions performed by the main operating system or those functions performed by the entry routines. The word entry technique for the program in Appendix C utilized the interrupt method; however, the interrupt service routine developed could be converted to a subroutine and used for a polling entry method. The segregation of the different functions is shown in Table X. This segregation

TABLE IX

PROCESSOR FUNCTIONAL SEGREGATION

Input--Processor #1	Network--Processor #2
Initialization of Devices	Initialization of Devices
Convert to Network Character Set	Store Information
Store Information	Format According to Network Protocol
Identify as Ready to be Processed	Transmit Information to Network
Check for Deletion of Local Information	Determine Routing
Identify as Ready to be Transmitted	Initialize Transmitter
Initialize Transmitter	Identify Information as Sent
Remove Network Protocol Information	Store Information
Transmit Information to Local Receiver	Determine if Error-Free
Recognize End of Message	Deallocate the Storage Space
	Process Information from Network
	Identify as Ready to be Transmitted
	Identify Type of Message
	Process Control Information
	Recognize End of Message

TABLE X

INPUT PROCESSOR OPERATING SYSTEM
FUNCTIONAL SEGREGATION

Main Operating System	Service Routines
Initialization of Devices	Convert to Network Character Set
Determine if Local Terminal Busy	Store Information
Initialize Transmitter	Check for Deletion of Local Information
	Recognize End of Message
	Identify as Ready to be Processed
	Identify as Ready to be Transmitted
	Deallocate Message Storage Space
	Transmit Information

assumed the processor stored the word, thus it was more efficient to perform certain functions upon word entry as opposed to later fetching the word from memory to perform these functions.

Initialization of Input Processor Operating System. The first part of the software design effort was involved with initialization. A composite SA diagram was used to functionalize the initialization process. This composite SA diagram is shown in Figure 5-1. The initialization consisted of two phases, a device phase and an operational phase. In the device phase, the different components on the processor card and the input card were programmed to their desired operational configuration. In the operational phase, the queues and tables needed for the proper operation of the universal network interface device were initialized.

Device Initialization Phase. The method used to initialize the I/O ports was based upon the idea of a linked list (Ref 36:71). Each of the 2651s and the processor board USART were required to have an associated parameter list. The content of the parameter list was dependent upon whether the I/O port was used in the synchronous mode of operation or the asynchronous mode of operation. For the asynchronous case, the parameter list consisted of the following:

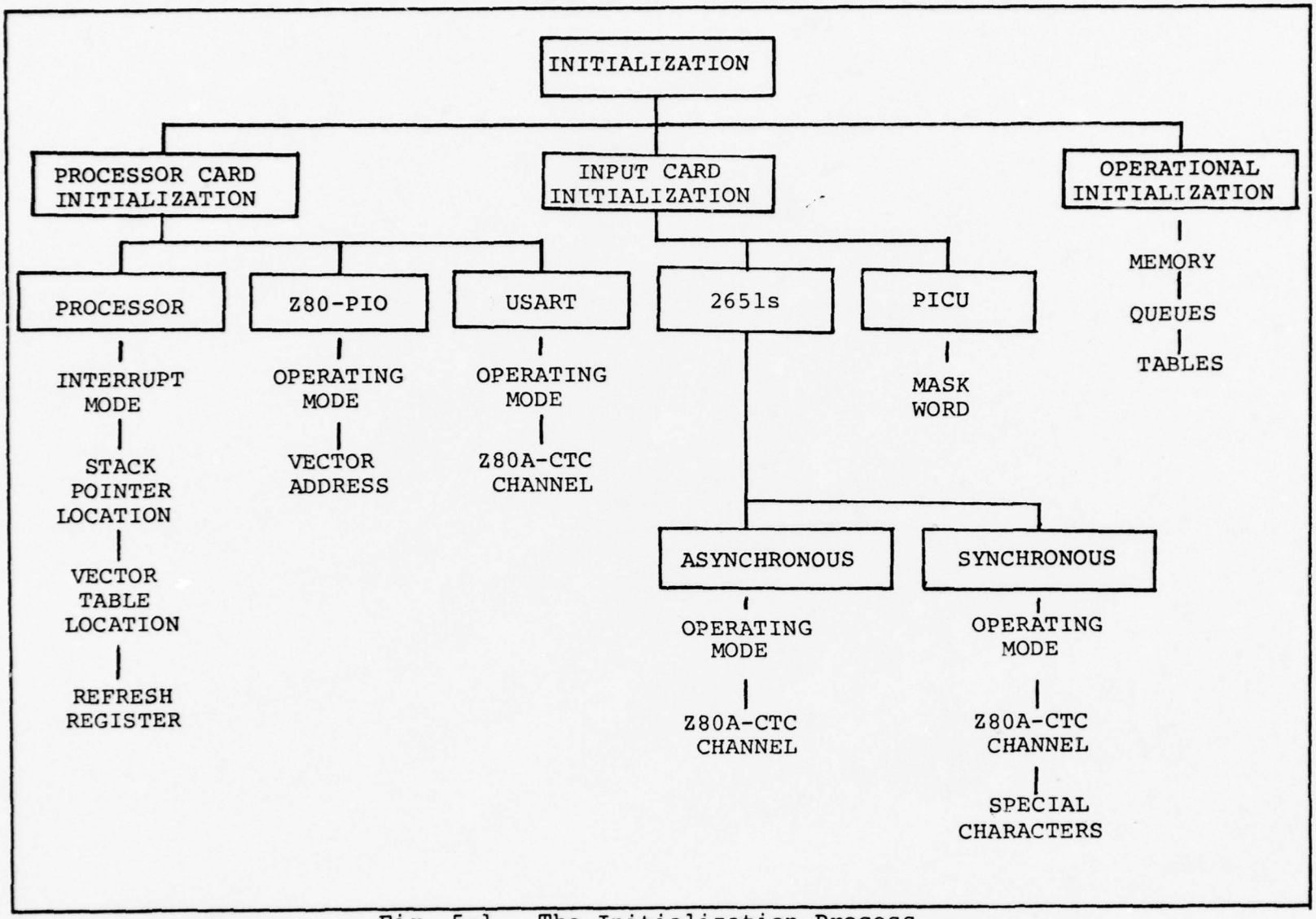


Fig. 5-1. The Initialization Process

list identifier	I/O address of the hold register
	Address of the location used to store the memory block address for a local message
	Word to be transmitted to the command register
	Word to be transmitted to mode register #1
	Word to be transmitted to mode register #2
	I/O channel address for the Z80A-CTC which supplied the frequency to the 2651
	Word to be transmitted to the Z80A-CTC mode register
	Word to be transmitted to the Z80A-CTC prescaler register
	Address of the next asynchronous 2651 parameter list

All of the asynchronous parameter lists were thus linked together and could be initialized with a looping section of code. A subroutine called ITUART within the loop actually accomplished the initialization. The flowchart for ITUART is shown in Figure 5-2.

The 2651s used in a synchronous mode of operation were initialized in a similar manner. Each of the synchronous 2651s had parameter lists which were linked together. The parameter list consisted of the information contained in the asynchronous parameter list plus three additional entries. These entries were the first synchronous character, the second synchronous character and the delete character. A looping section of code was used to initialize all of the synchronous 2651s within the linked list. The subroutine ITUART was used to output the first section of

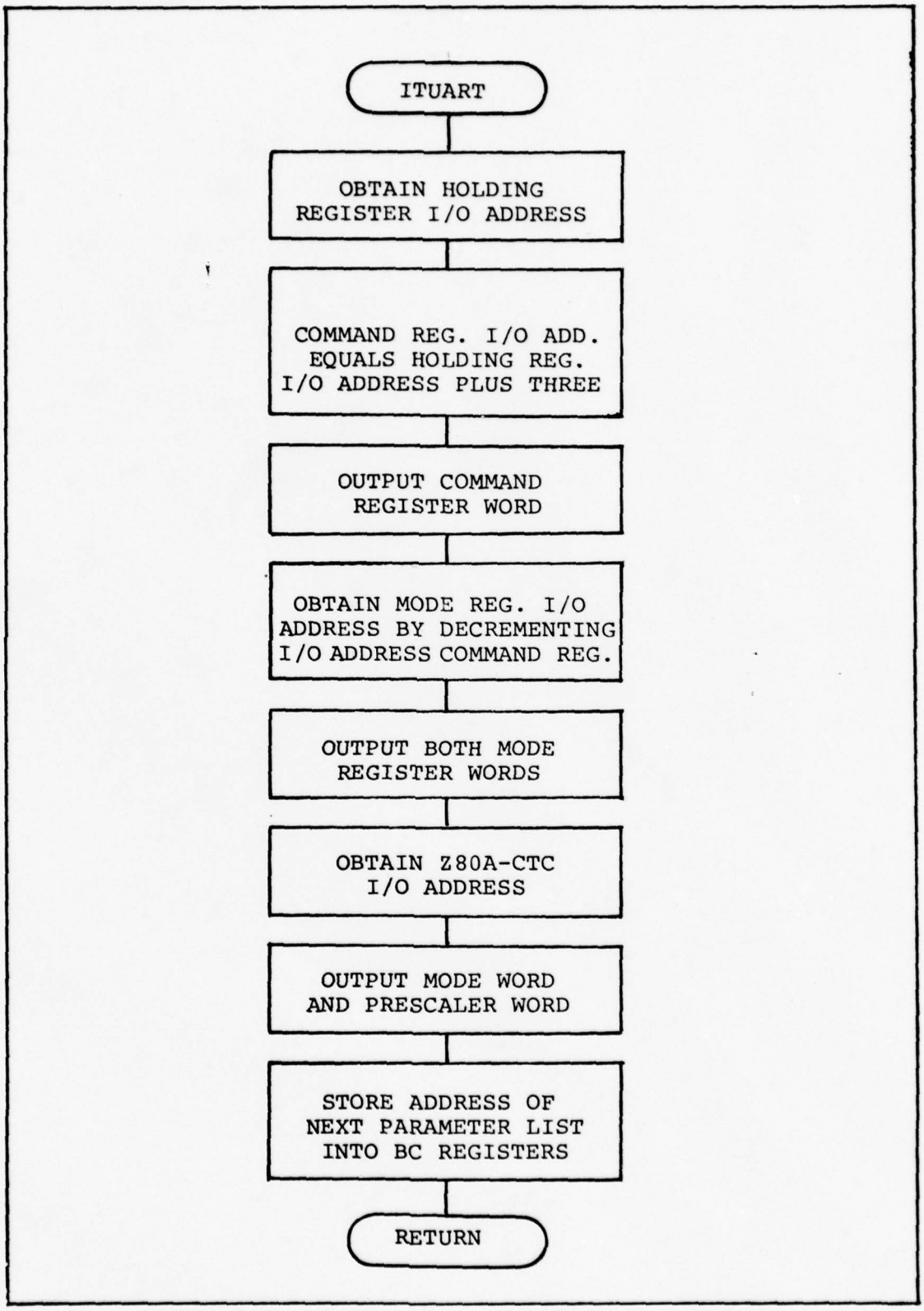


Fig. 5-2. Subroutine ITUART Flowchart

information with the latter three entries outputed after the return from subroutine ITUART.

The next group of chips which required initialization were the priority interrupt controllers. These were again organized into a parameter linked list structure. The parameter list contained the following:

list identifier
I/O address of PICU
The mask word to be outputed
Address of next parameter list

The two chips still requiring initialization were the processor chip and port A and B of the Z80-PIO. Each of these were initialized with an individual section of code.

Operational Initialization. The operational initialization phase involved the initialization of the different queues and tables required for operation of the universal network interface device. From Table X, it was established that three queues would be required for use by processor #1. A network transmit queue (NWTXQ) was needed to transfer from processor #1 to processor #2 the storage address of those messages to be transmitted on the network. Conversely, a local transmit queue (LOTXQ) was needed to transfer from processor #2 to processor #1 the storage address of those messages to be transmitted to peripherals connected to the inferface. A third queue (LBTXQ) was needed by processor #1 to store the memory address of messages which could not be transmitted to local peripherals because the peripheral was still receiving a previous message.

Each of the queues were designed to be circular in nature with two 16-bit locations used to control queue operation. These 16-bit locations contained the address of the current head of the queue and the address of the current tail of the queue. The queue initialization consisted of setting the head and tail of the queue to the address of the start of the queue.

The other operational initialization requirement was established by the store information function. Within the function was a requirement to determine where an incoming message would be stored. This suggested a table be constructed which consisted of the memory addresses of all unused memory. As the memory was used, it would be removed from the table. It would be put back into the table by the deallocate storage space function. To allow this table to be generated internally required certain information and assumptions be made about the memory structure. First, it was assumed a large contiguous section of memory would be dedicated to message storage. This section would then be broken up into a number of fixed sized memory blocks which would be allocated through the memory table. The initialization routine generated the memory table based upon the value associated with certain variables. The values required were the address of the start of the memory table (LOMNTB), the address of the start of the contiguous section of memory (MENST), the number of memory blocks (BLKNUM) and the size of each memory block (BLKSIZ). The

maximum block size was limited to 256 to simplify the operations associated with this value.

Operating System #1 Generalized Subroutine

This concluded the initialization portion of processor #1. The initialization generated a requirement to add/delete memory addresses from different queues and from the memory table. The next section discusses the generalized design of such routines.

Queue Addition/Deletions. The operations associated with a given queue were limited to the addition of a memory block address at the tail of the queue and the removal of the memory block address from the head of the queue. In a network application, there may be a method to identify an important message which would allow it to be added to the head of the different queues. A routine to do this was not included in the operating system. If required, this routine could be easily developed as it would be a slight variation of the other routines. The lock-out method developed for jointly shared queues would support this other routine.

When the need for the different queues was discussed, the information within the queues was shared and changed in two instances by both processors. There could arise a situation where one processor was changing information while the second processor was reading this same information. Thus, entry to the information in the shared queues

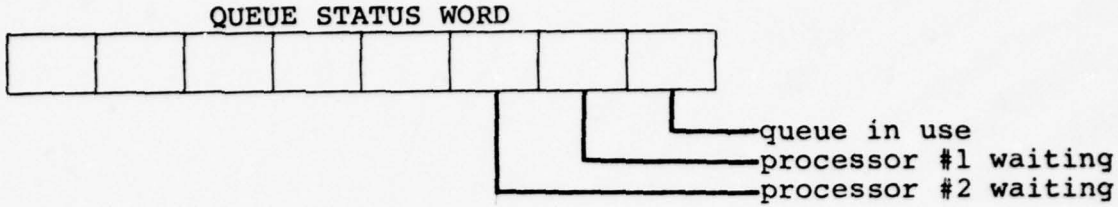
had to be controlled to insure only one processor had access to the queue at a given point in time.

The method chosen for control of the queues was through the use of a queue access word. The processor desiring access to the queue would test a bit to determine if the other processor was using the queue. If not, it would set a bit in the access word to reflect it was using the queue. While this approach was feasible, it could not be directly implemented. A problem resulted because of the instruction execution relationship between the two processors. When one processor attempted to gain access to information in a queue, the other processor could be from one-half clock cycle to any multiple thereof of also attempting to gain access. If the two processors were within a half clock cycle of each other, both would test for the other, determine the queue was free, set if queue status to using, and begin changing information contained within the queue. This clock relationship dictated a more sophisticated access technique be designed.

Since the processors could be so close in synchronization, a delay had to be introduced into the entry routine of one of the processors. Processor #2 was designated as having priority over processor #1 in the use of any of the queues. On attempting to gain entry to any of the shared queues, each of the processors would set unique bits to indicate it was waiting for the queue. They would then test to determine if the other processor was waiting. If

so, processor #1 would jump into a loop, while processor #2 would determine processor #1's status concerning use of the queue. This was accomplished by testing another bit to determine if the queue was in use. If not, processor #2 would set the bit indicating it was using the queue and proceed with its action. If processor #1 was using the queue, processor #2 would be put into a wait loop until processor #1 was through with the queue. The actual code for this is shown in Figure 5-3 with an execution timing diagram shown in Figure 5-4. The timing diagram (case 1) shows that for the case of O.S. #2 attempting to gain access to the queue ahead of O.S. #1 the lock-out code would function properly. Case #2 illustrates the worst case for the situation when O.S. #1 is ahead of O.S. #2 in terms of queue access actions. In this case, O.S. #1 tests the waiting status of O.S. #2 one-half clock cycle before it is changed. Again the lock-out code functions properly as the bit 0 instruction would cause O.S. #2 to be put into a wait loop.

Once the lock-out mechanism was designed, the flowchart for the queue addition and deletion tasks were developed. These are illustrated in Figures 5-5 and 5-6. To insure these algorithms work properly, the queue must consist of an even number of locations with the following structure:



O.S. #1 Code

```
Loop    LD HL, address of queue status word
        SET 1, (HL)    ; set status word to processor #1 waiting
        BIT 2, (HL)    ; check if processor # 2 waiting
        JP NZ, LOOP    ; loop if processor # 2 waiting
        SET 0, (HL)    ; set status word to O.S. #1 using
```

O.S. #2 Code

```
        LD HL, address of queue status word
        SET 2, (HL)    ; set status word to processor #2 waiting
        BIT 1, (HL)    ; check if processor #1 waiting
        JP Z, QFREE    ; jump if processor #1 not waiting
Loop    BIT 0, (HL)    ; check if processor #1 using
        JP Z, LOOP     ; loop if processor #1 using
QFREE   SET 0, (HL)    ; set status word to processor #2 using
```

Fig. 5-3. Processor Lockout Mechanism

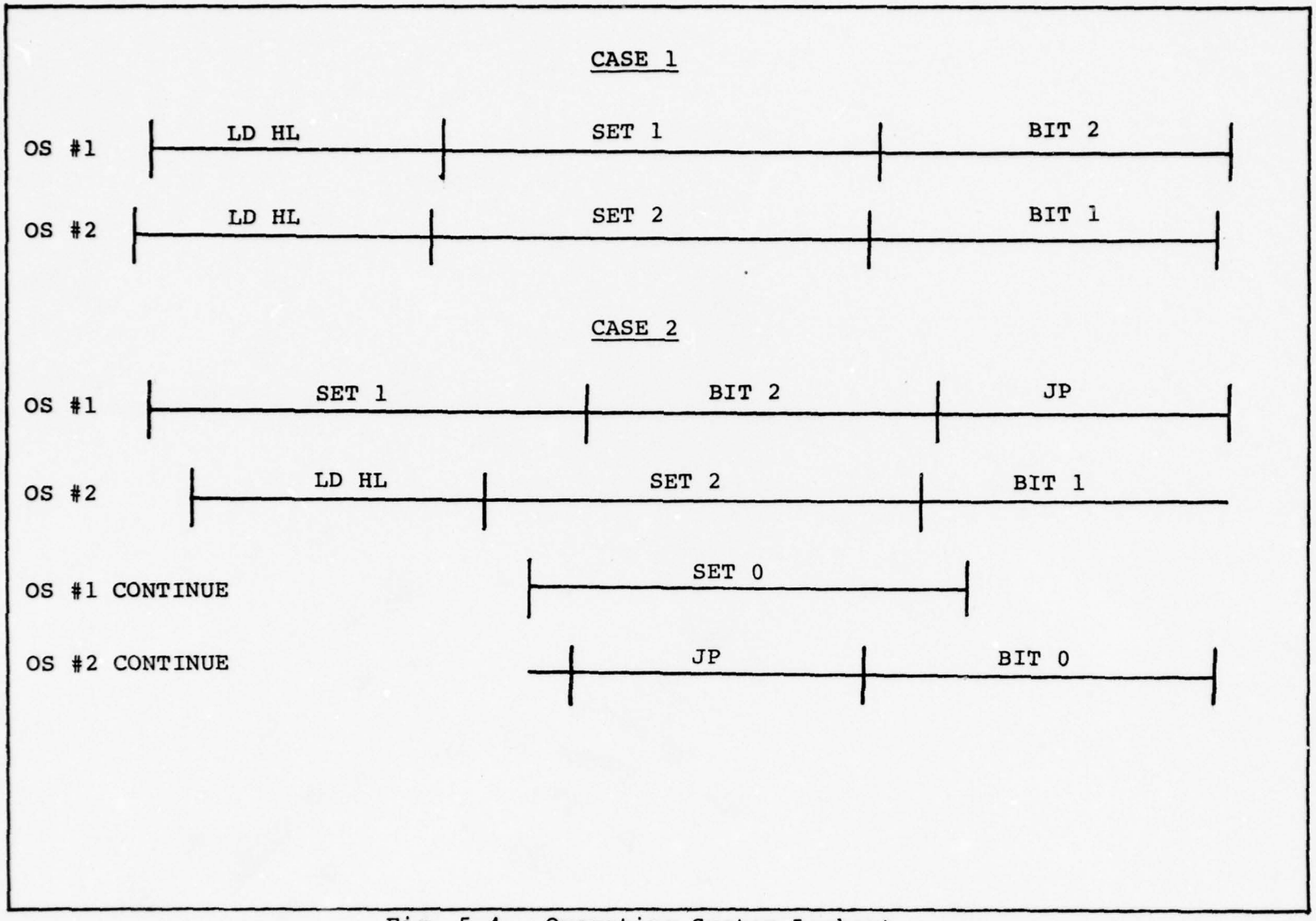


Fig. 5-4. Operating System Lockout

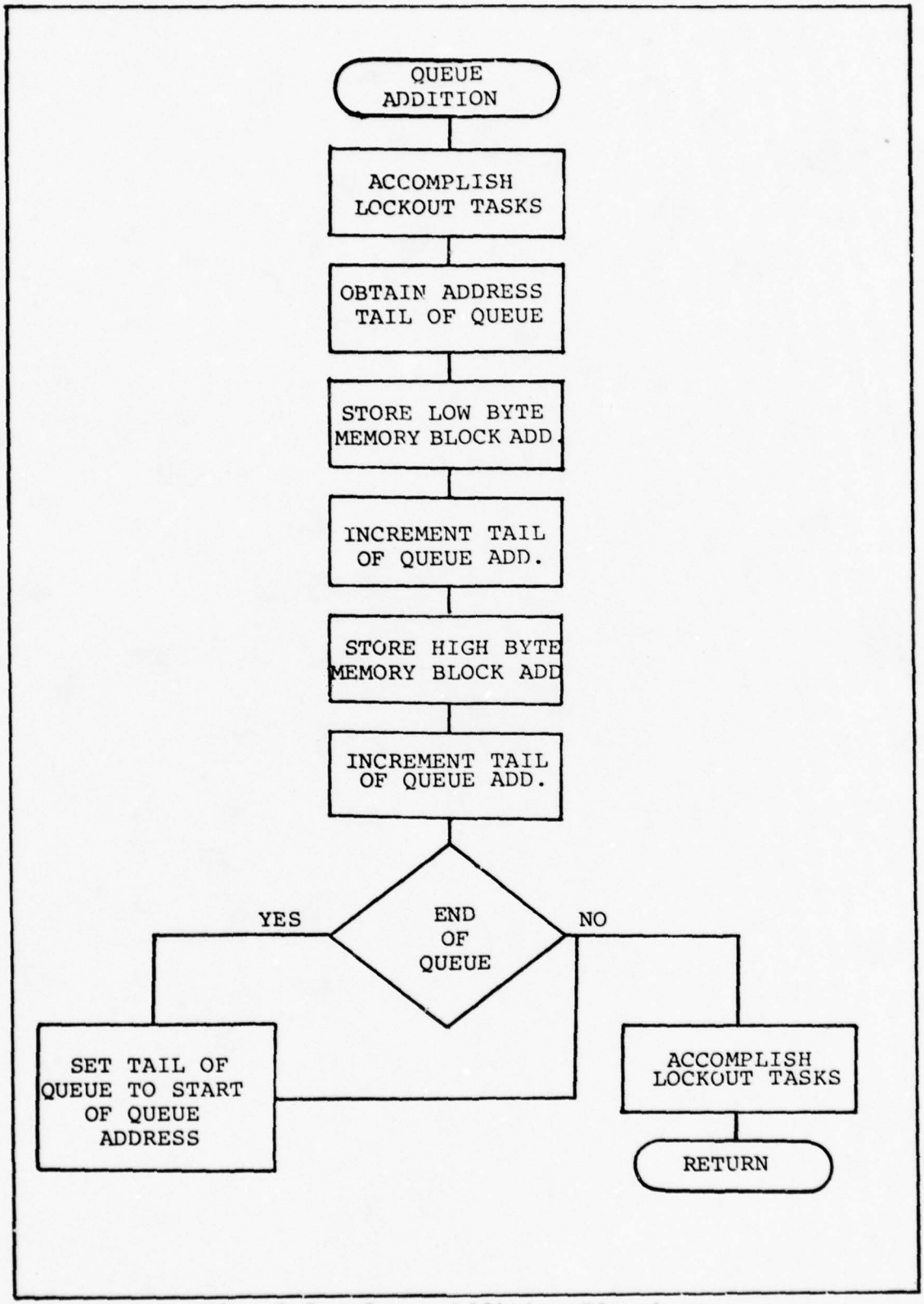


Fig. 5-5. Queue Addition Flowchart

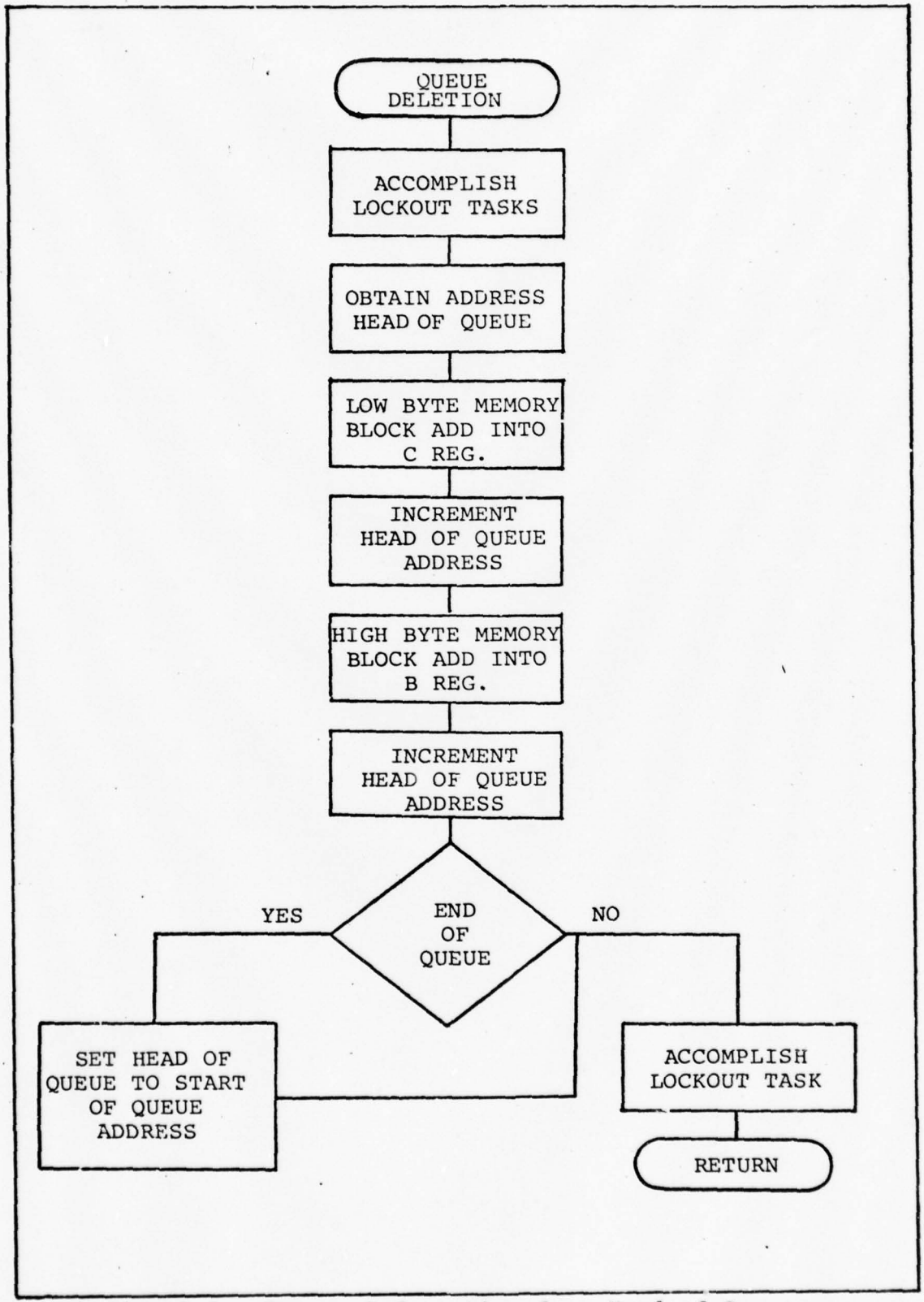


Fig. 5-6. Remove Information from Head of Queue

Start of queue address	XXXXXX
	XXXXXX
	.
	.
	.
	.
	.
	XXXXXX
End of queue address	XXXXXX

This structure was chosen to reduce processing time associated with the end of queue check. The sixteen-bit subtract instruction which must be used to make this check includes a carry flag subtraction. With this structure, the proper result is obtained irrespective of the value of the carry flag.

Memory Table Addition/Deletion. The actions required to be accomplished on the memory table were similar to the shared queue actions. Since the memory table would be used by both processors, the lock-out code would be required for both actions. The memory table was set up with only a head pointer. This was done to eliminate an end of table check for the addition action. The required actions consisted of addition to the head of the table and deletion from the head of the table. The addition algorithm was very simple and is not presented in this paper. The deletion action was similar to the remove information from head of queue algorithm except that after the lock-out tasks were accomplished, a check had to be made to determine if memory was available. If memory was not available, a wait loop was entered, until a block was freed. This approach was used

since memory was allocated upon receipt from the user peripheral of the first character. In an actual application, a more formalized local protocol procedure could be used which required the peripheral to obtain access to the universal network interface device before sending a message. The right to access would then be conditioned upon whther memory was available or not. The deletion flowchart is shown in Figure 5-7.

Interrupt Service Routines
Operating System #1

The next routines developed were those routines which would normally be used to service a teletype or CRT terminal connected to the universal network interface device. Within the routines, certain simplifying limitations were imposed to reduce the complexity of the code. The address information provided to the universal network interface device was limited to two characters. The first character was the destination address while the second character was the sender address. Thus terminal identifications were limited to zero through none or A through Z. This was done to minimize the development of a local protocol for the testing of the universal network interface device. By limiting the address, conversion and packing of multi-character address was not required. To send a message, all the terminal had to do was to begin typing the destination address of the message.

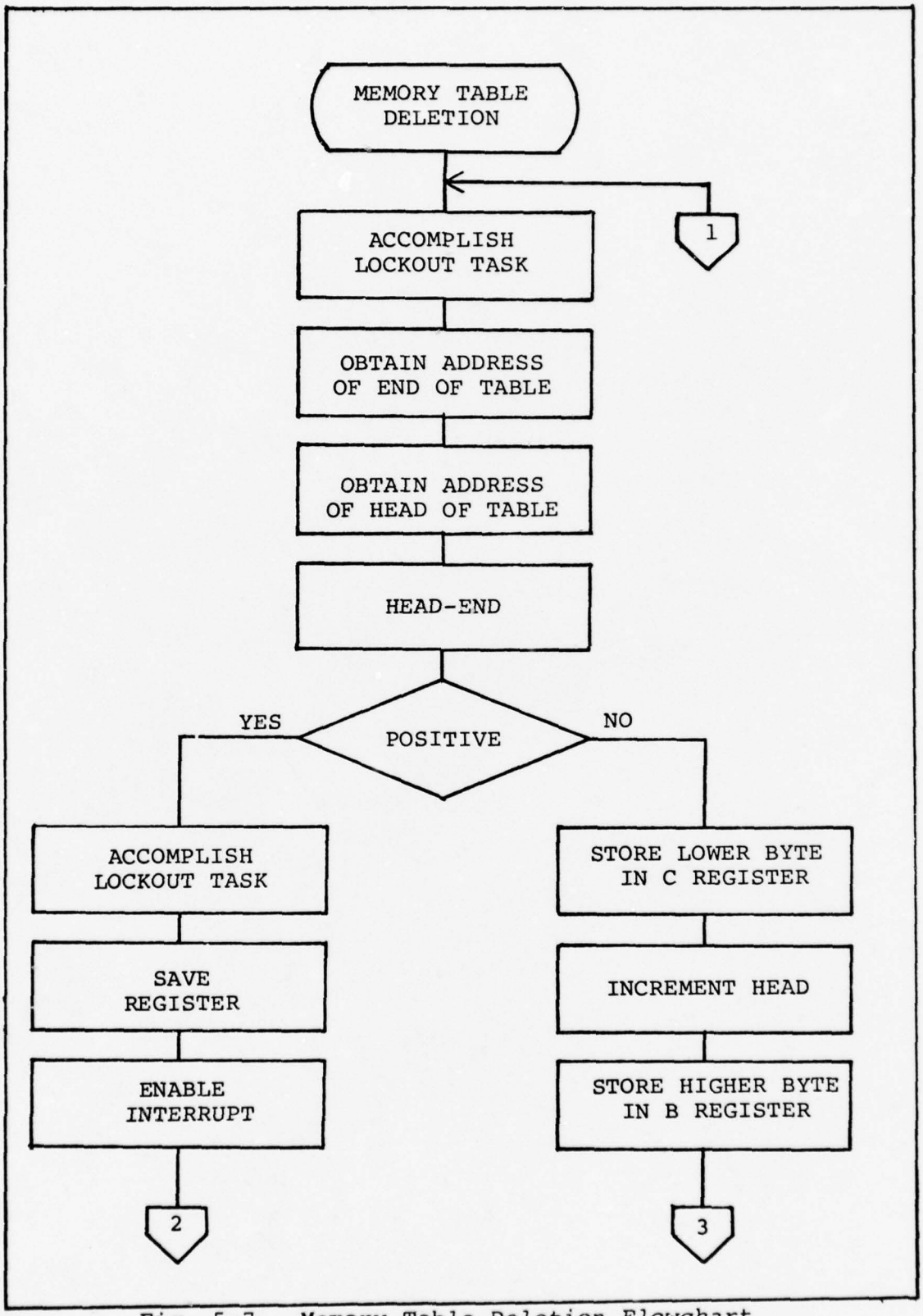


Fig. 5-7. Memory Table Deletion Flowchart

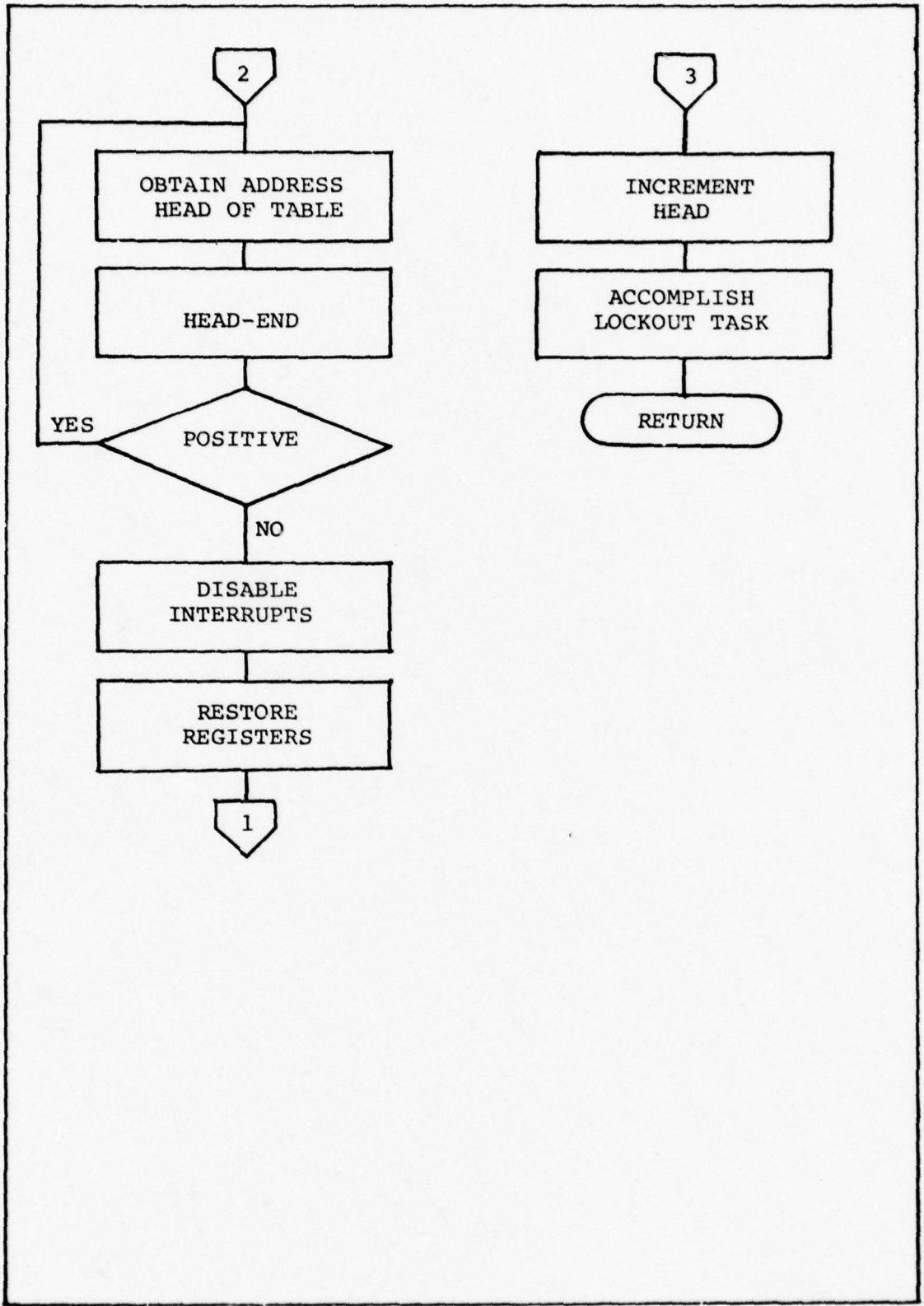


Fig. 5-7--Continued

Operating System #1 Receive Interrupt Routine. This subroutine implemented for a TTY or CRT terminal the receive functions under the service routine breakout in Table IX. The routine would normally be entered upon generation of a character bit stream by the terminal. The character bit streams would continue to be stored until an end of message character was received. Upon receipt of this character, the memory block storage address would be added to the tail of the network transmit queue for further processing by processor #2.

In development of the algorithm for this routine, the situation where a message length exceeded the memory block size was considered. One approach was to link the memory blocks and then transmit the complete message after it was received. However, the message packet transmission concept is gaining increasing support as an efficient method of message transmission. If the memory block size was defined to equal the maximum packet size, then a packeting concept could be implemented. Counter to other decisions which simplified the code, the latter concept was selected as the method to handle message block storage overflow. To implement this method required a control word be sent with each message. The control word was organized as shown in Figure 5-8. A one in bit position four signified the message was one packet in a sequence of packets. A one in bit position five signified the message was the end packet of the sequence.

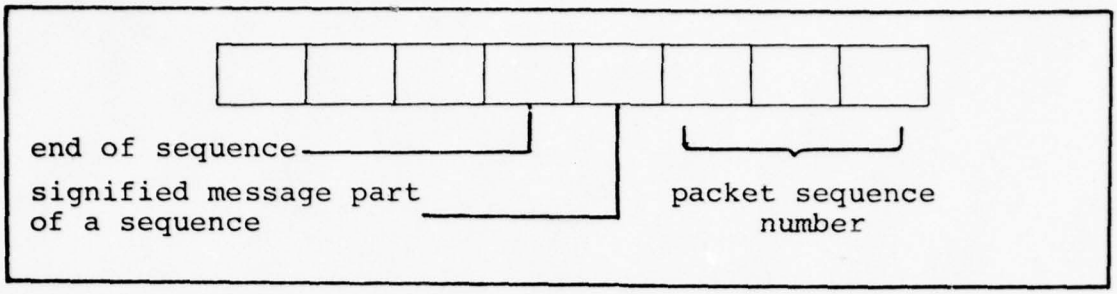


Fig. 5-8. Packet Sequence Control Word

The flowchart for the receive service routine is shown in Figure 5-9. To support this routine required five eight-bit words be allocated for use by the service routine. These words were used to store the address of the memory block allocated to the routine, the current number of words stored in the memory block, and the message control word.

Operating System #1 Transmit Interrupt Routine. The transmit interrupt service routine implemented the transmit functions under the service routine in Table X. It transmitted a word of information in response to a transmit buffer empty interrupt. The flowchart for the routine is shown in Figure 5-10. To implement this routine required eight eight-bit words be allocated for use by the routine. These words were used as follows:

Words 1 and 2	Address of memory block being transmitted
Words 3 and 4	Multibuffer address of next memory block
Words 5 and 6	Address of multibuffer status word
Words 7 and 8	Number of words transferred

The multibuffer address is the address of a portion of memory used to assemble the packet sequences of a message. There can be any number of these multibuffer storage areas in memory. They require 19 contiguous storage spaces

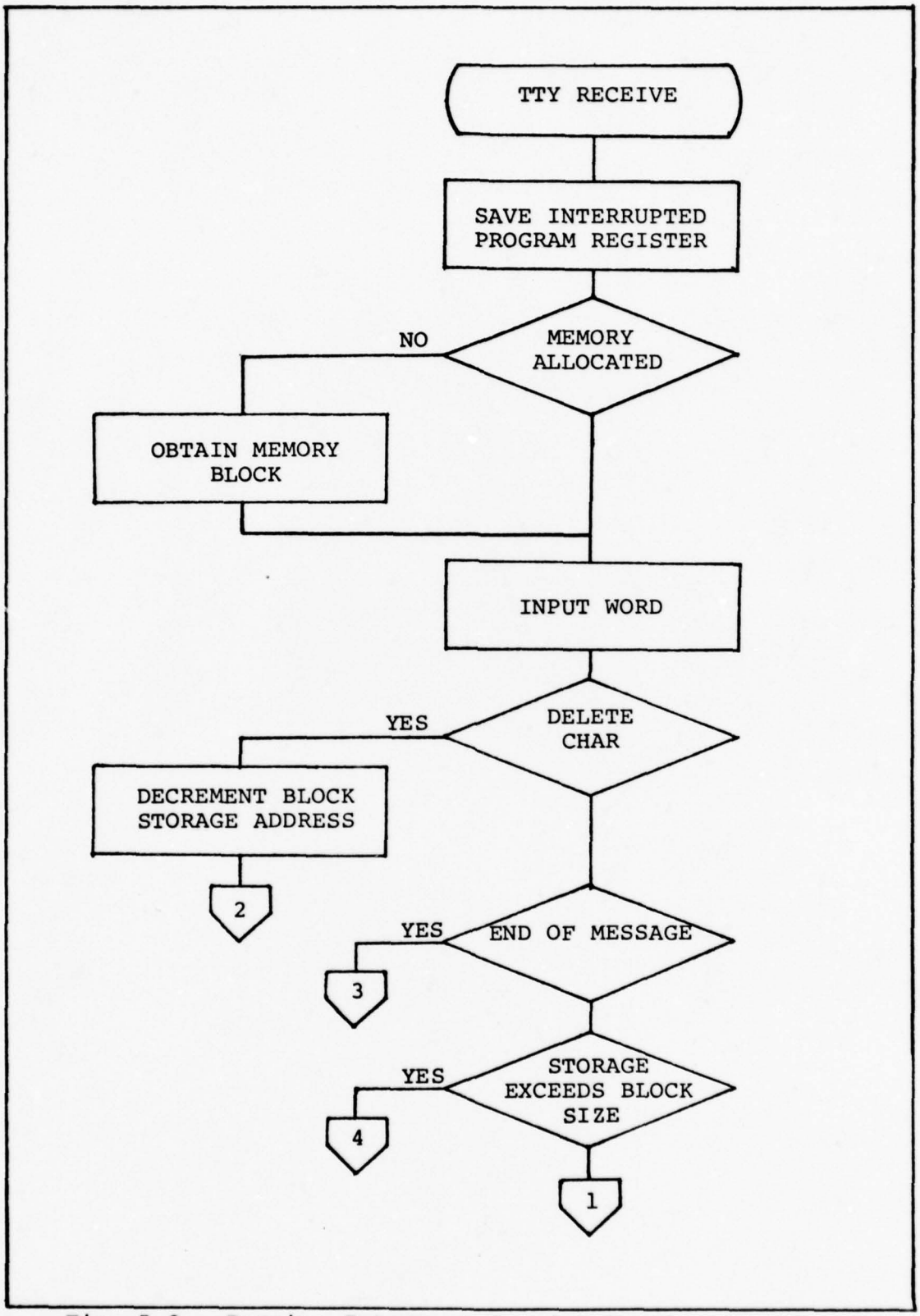


Fig. 5-9. Receive Interrupt Service Routine Flowchart

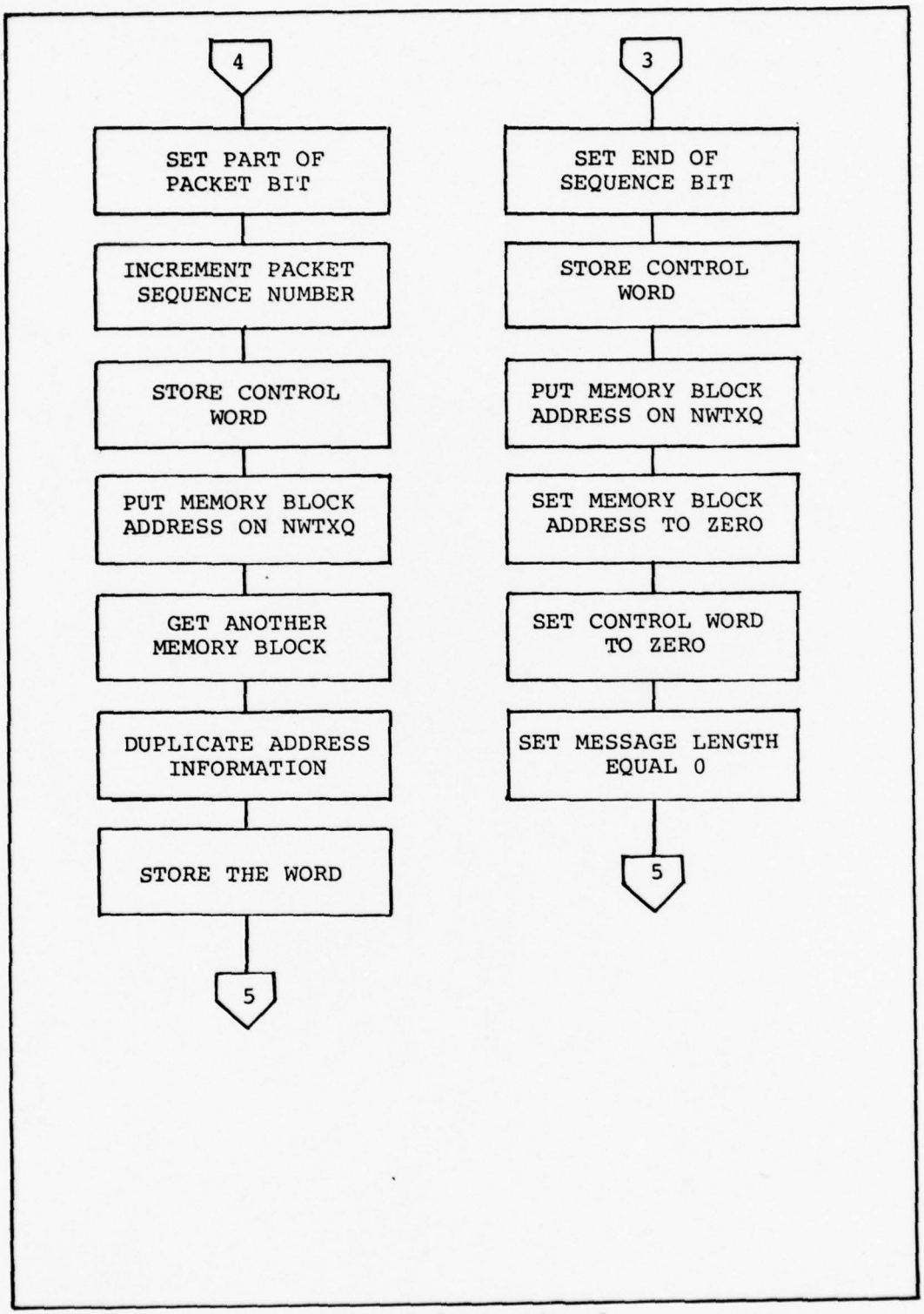


Fig. 5-9--Continued

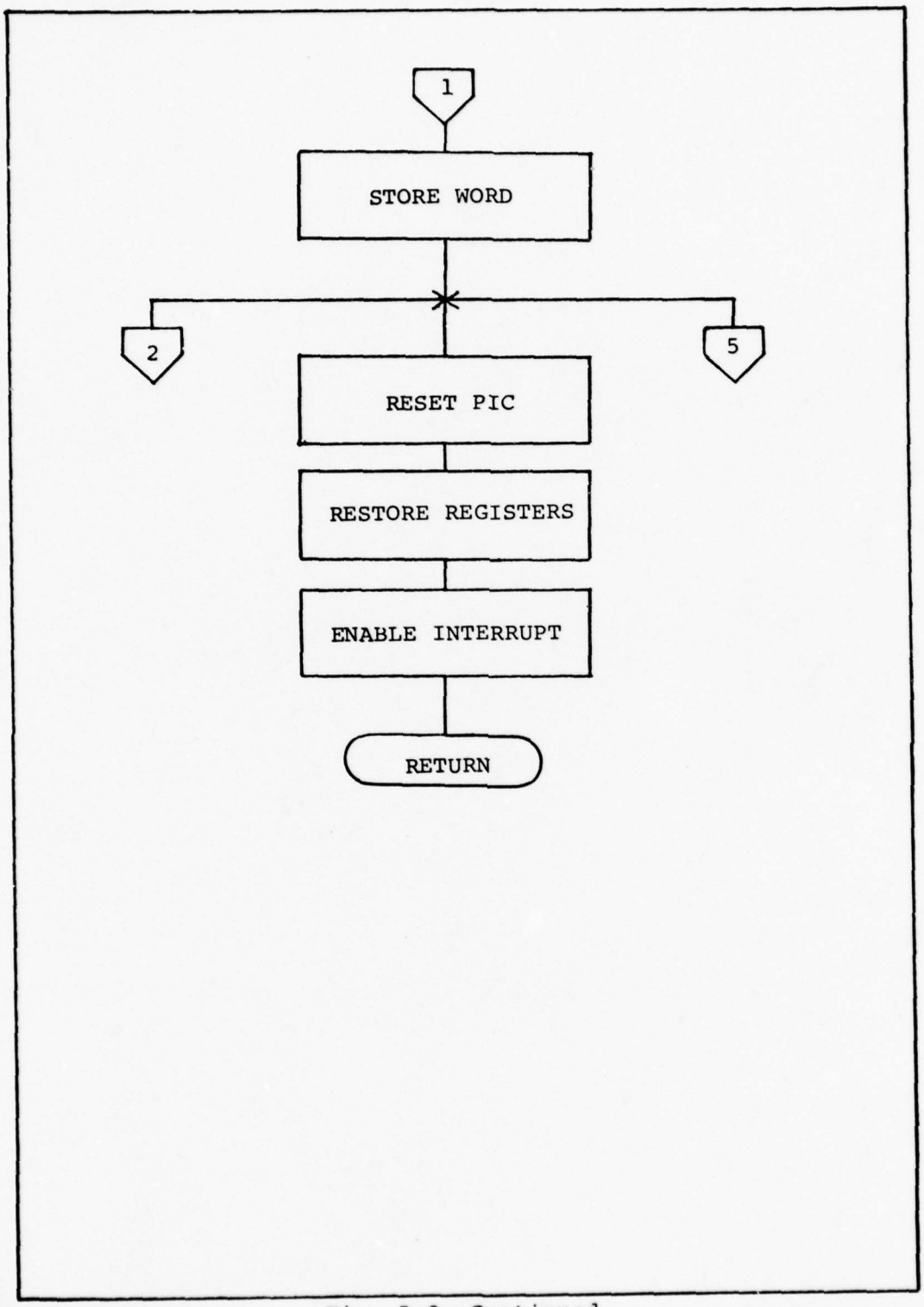


Fig. 5-9--Continued

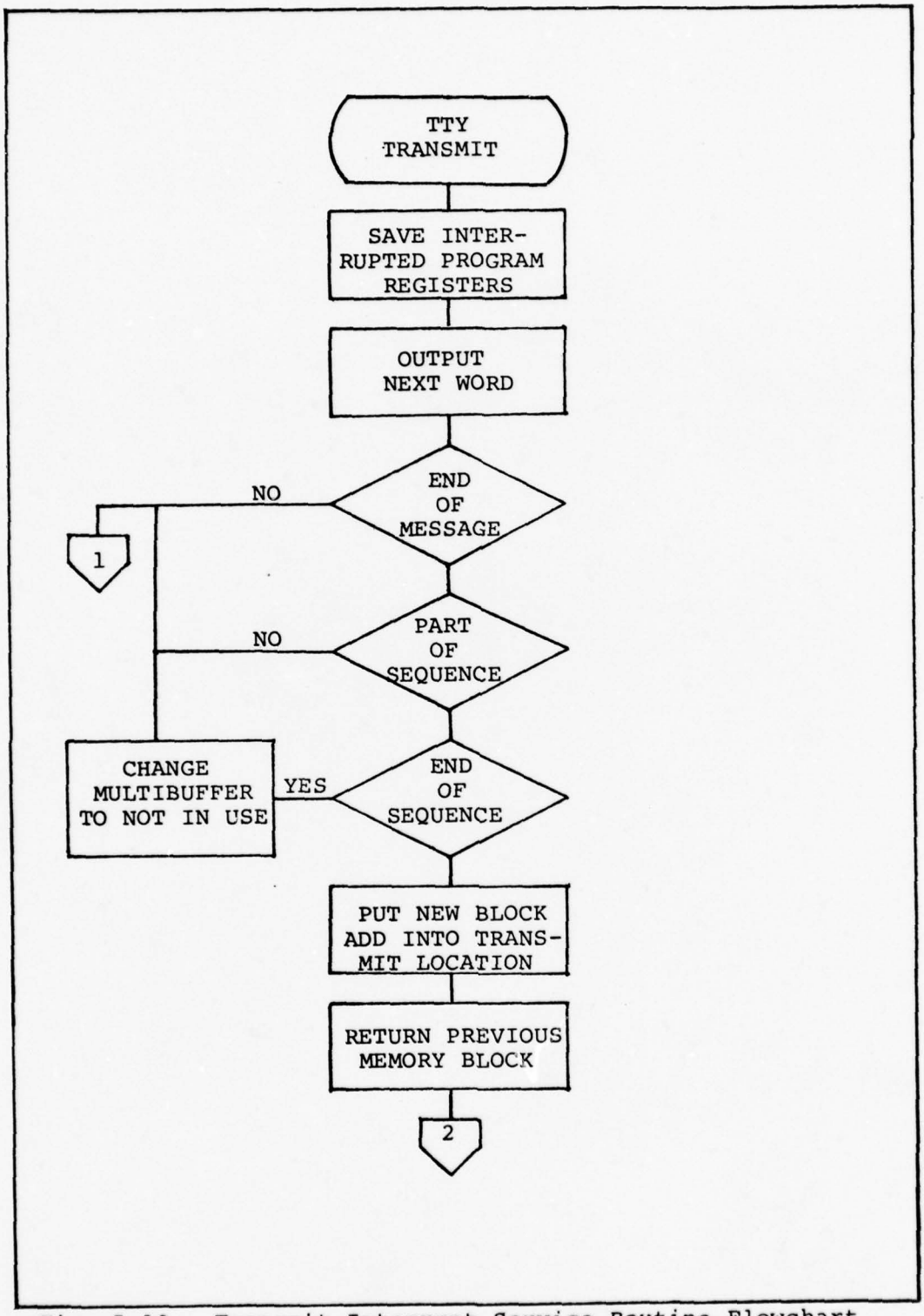


Fig. 5-10. Transmit Interrupt Service Routine Flowchart

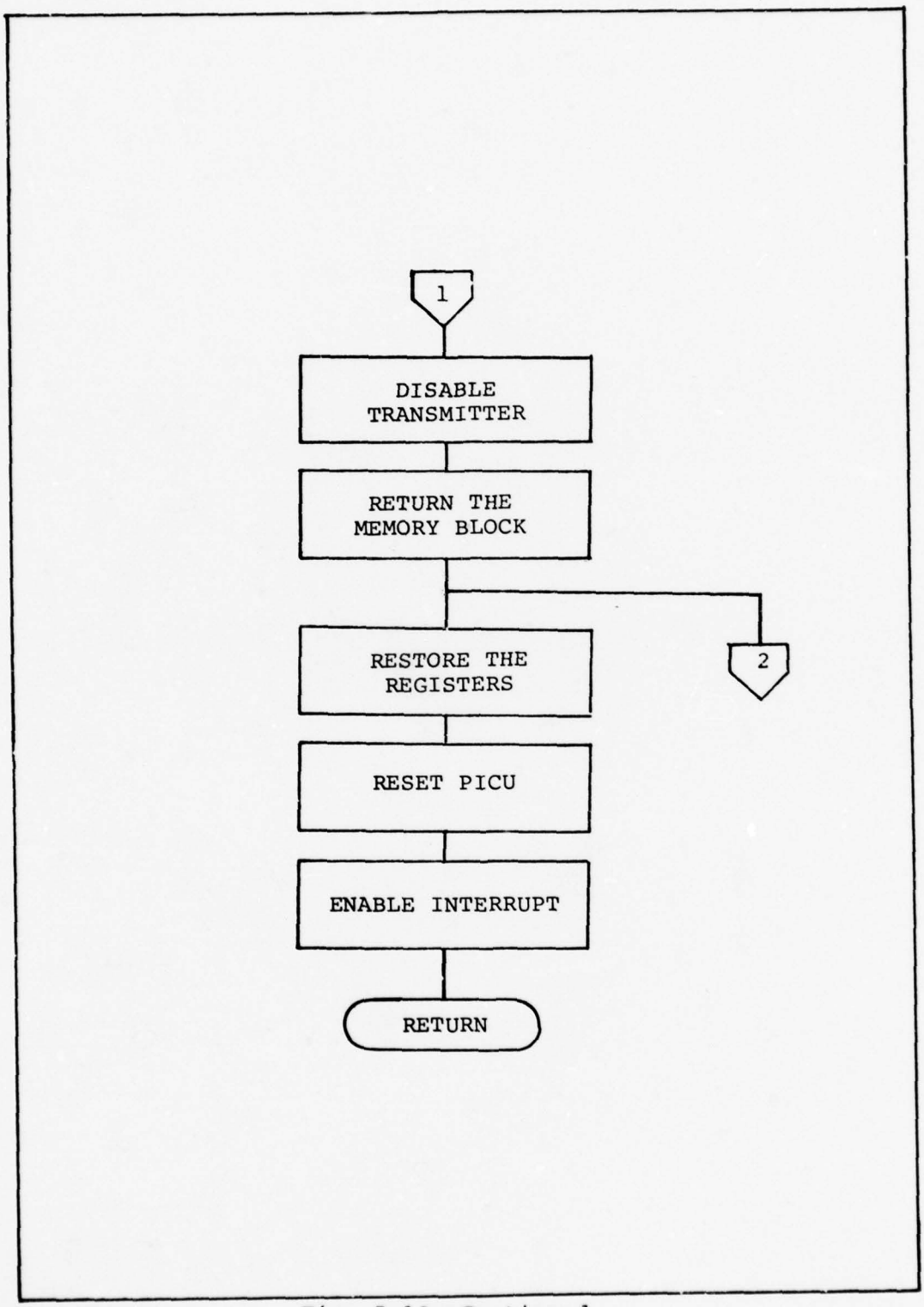


Fig. 5-10--Continued

which are used as follows:

Word 1	Status word for the assembly area
Word 2	I/O address of the holding register
Word 3 and 4	Address of location used to store the memory block address of message being transmitted
Word 5	Sender's address of the message
Word 6 thru 19	Addresses of the different blocks where the packet sequences are stored

The assembly area status word is organized as shown in Figure 5-11. This completed the design necessary to accomplish the service routine functions.

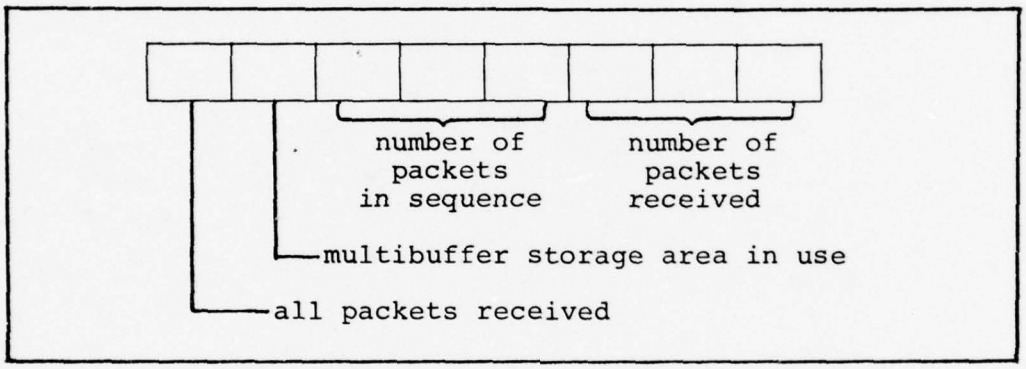


Fig. 5-11. Multibuffer Status Word

Main Operating System #1

The main operating system performed a monitoring function. It monitored the multibuffer storage areas, the local transmit queue and the local busy transmit queue and took action based upon certain conditions. These actions normally involved enabling of the transmit portion of a 2651 and the loading of the message memory block into the 2651 transmit address location. These functions correlated to those shown in Table X for the main operating system. These would have been the only functions of the main operating

system had the packet concept not been implemented. The packeting concept increased the complexity of the operating system because of the tasks associated with arranging the packets into the proper sequence. Once processor #2 put information into the local transmit queue, operating system #2 had to perform the tasks shown in Figure 5-12.

The network address is correlated to the local address through use of two tables. The first called the network address table (NWADTB) contains the network address of all peripherals connected to the universal network interface device. For each entry in the network table, there must be a corresponding entry in the local address table (LOADTB). The entries required in the local address table are the local I/O address which corresponds to the network address and the address of the location used by the service routine to store the memory block address of the message being transmitted.

The overall flowchart of operating system #1 is shown in Figure 5-13. This then completed the design of operating system #1.

Network Processor Operating System

The functions which are performed by processor #2 and operating system #2 are shown in Table IX. In a similar manner, these functions can be segregated into those functions performed by service routines and those performed by the main operating system. This segregation is accomplished in Table XI.

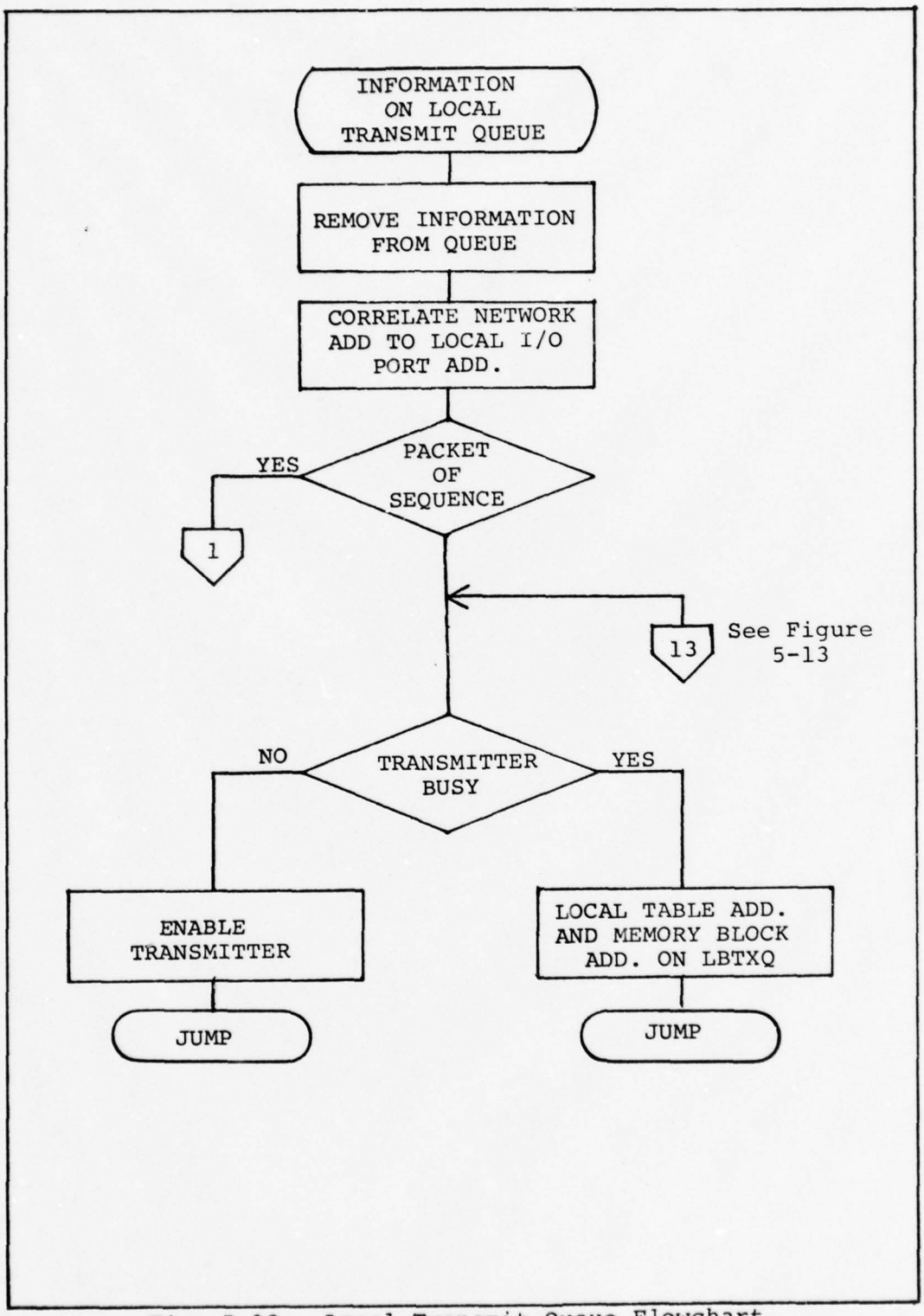


Fig. 5-12. Local Transmit Queue Flowchart

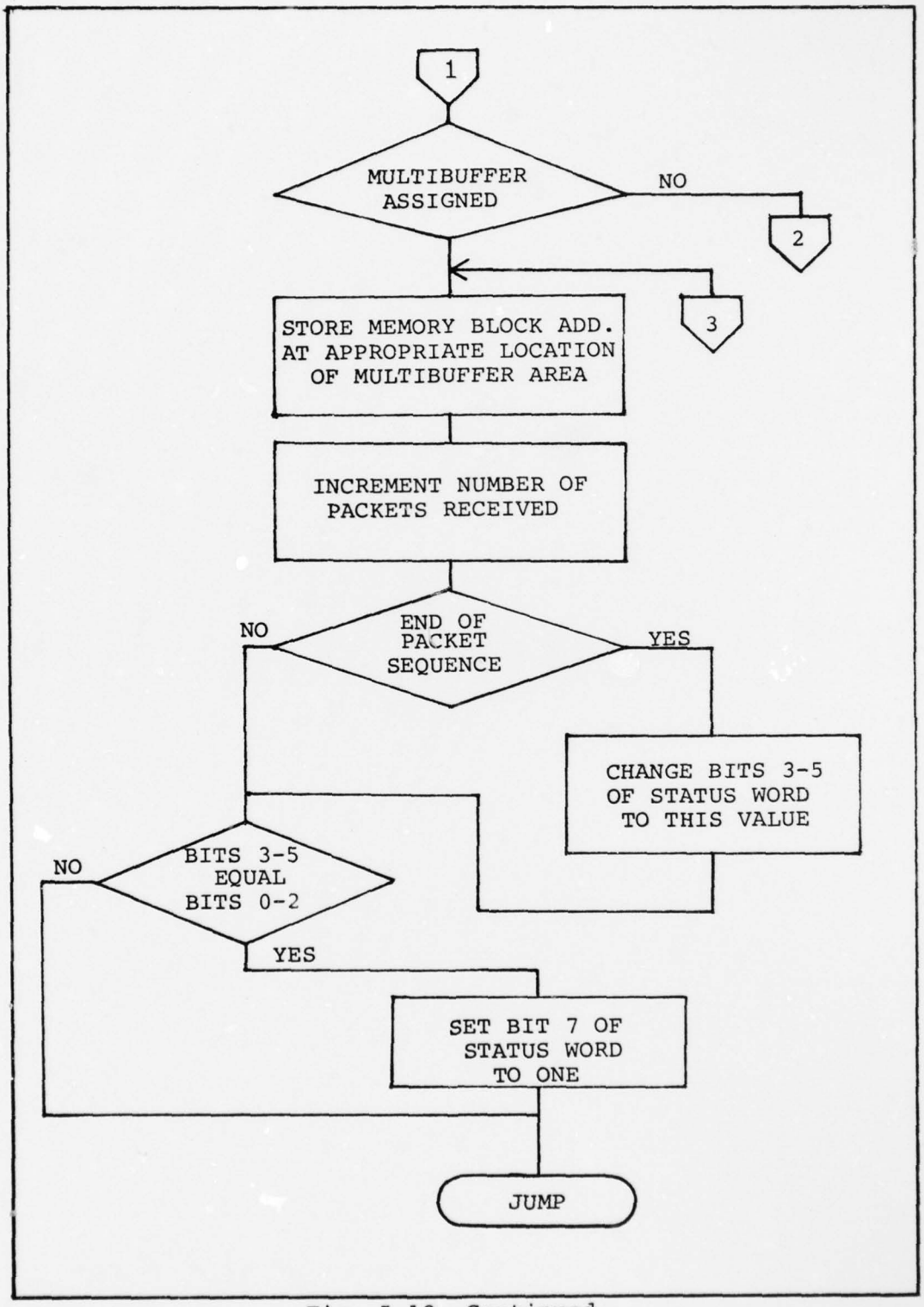


Fig. 5-12--Continued

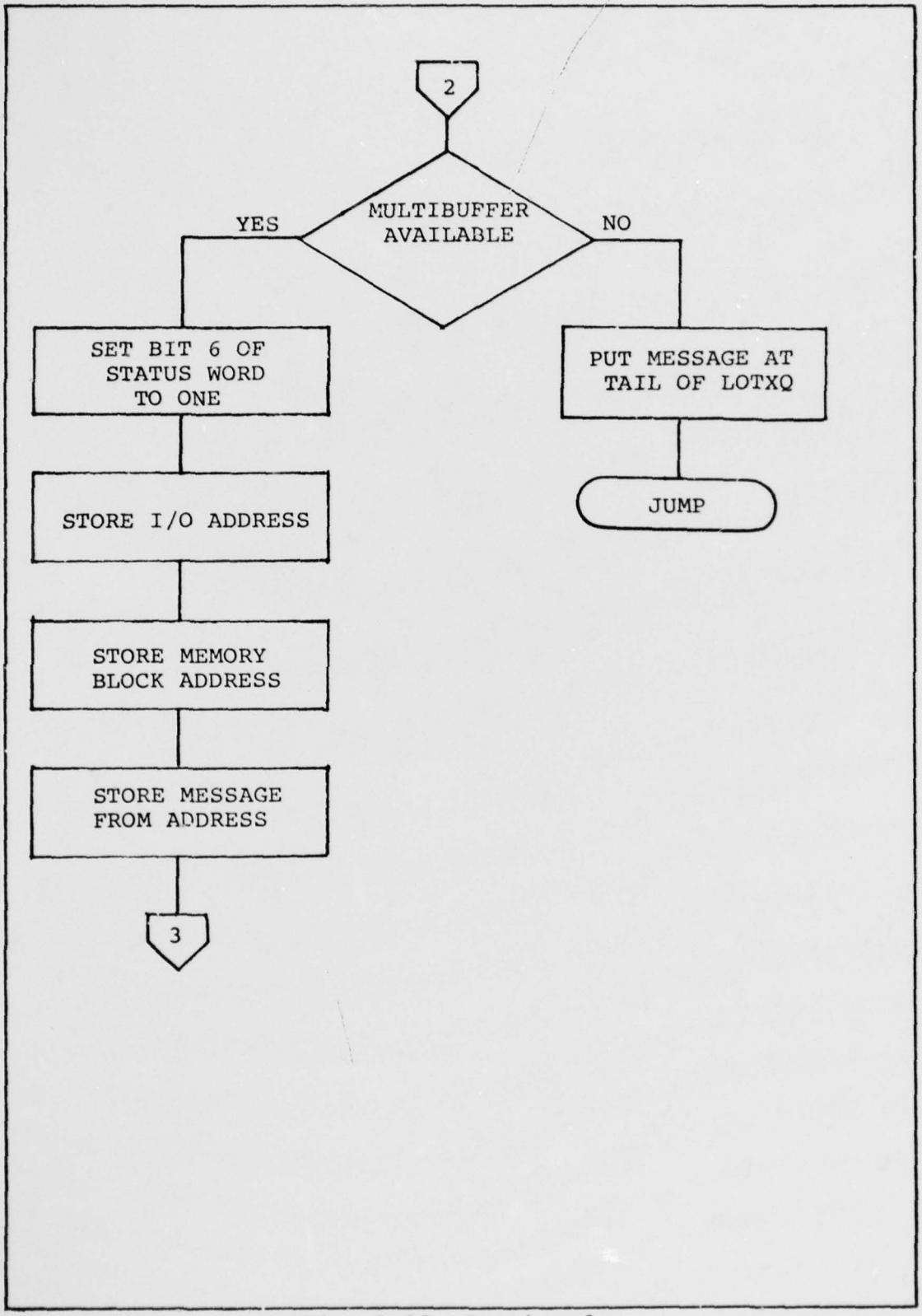


Fig. 5-12--Continued

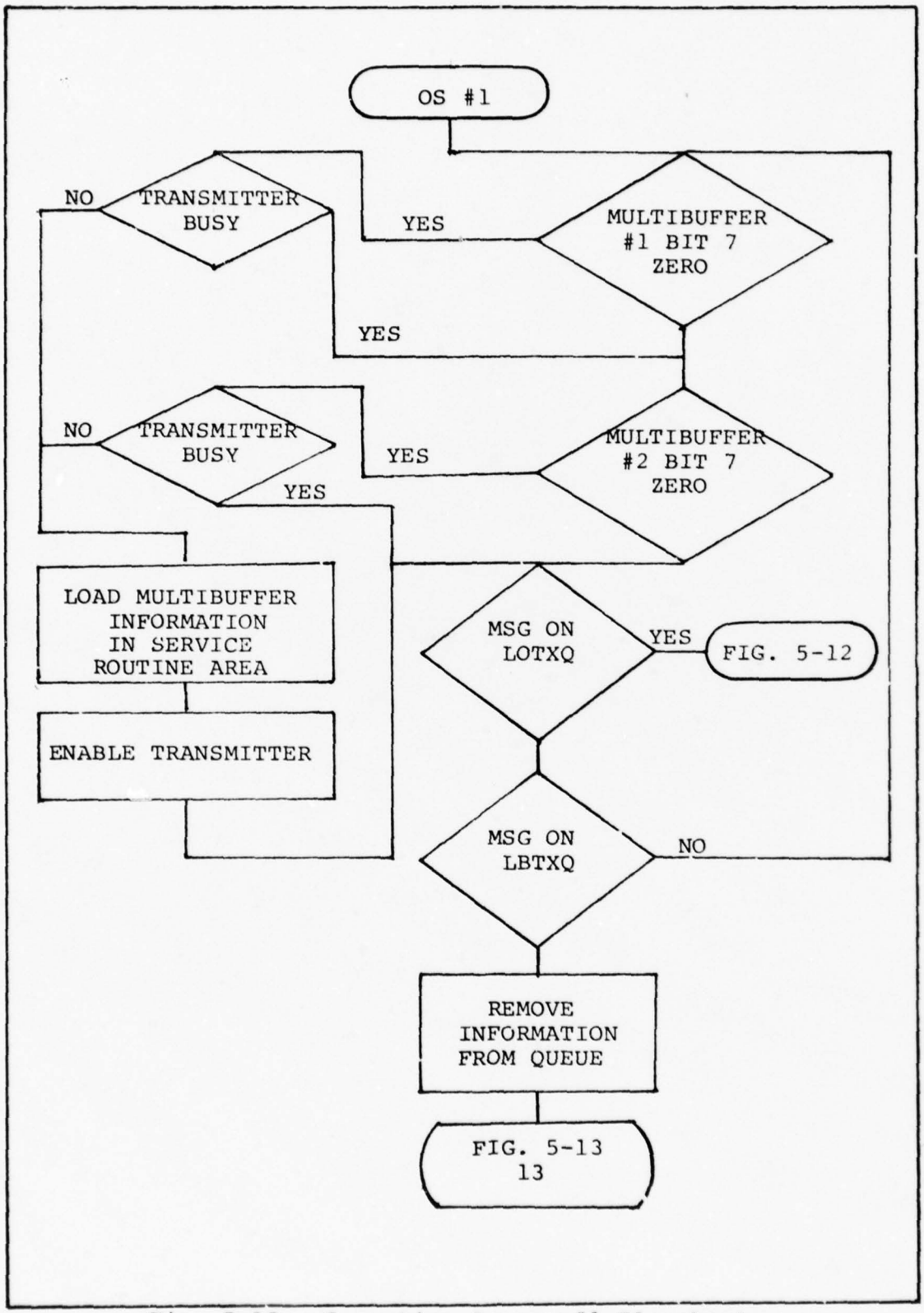


Fig. 5-13. Operating System #1 Flowchart

TABLE XI

NETWORK PROCESS OPERATING SYSTEM
FUNCTIONAL SEGREGATION

Main Operating System	Service Routines
Initialization	Store Information
Determine if Message for This Location	Determine if Error-Free
Generate ACK Message	Identify as Ready to be Processed Network Information
Identify as Ready to be Transmitted	
Determine Routing	Transmit Information to Network
Initialize Transmitter	Identify Information as Sent
Identify Type of Message	Set Message Timer
Process Control Information	Recognize End of Message
Remove Network Protocol Information	
Deallocate the storage space	

Initialization of Network Processor Operating System.

The initialization of processor #2 required devices be initialized. Device initialization was required for the processor, the processor board USART and associated chips, and the network card's Z80A-SIO and Z80A-CTC. Each of these initializations were done in an individual section of code. The Z80A-SIO and Z80A-CTC initialization did employ a parameter list and the capability to link the parameter lists together. For the Z80-SIO, the parameter list consisted of the following:

List Identification
- I/O address for Port A command
- Values for the different registers (Ref 28:12-20)
- Address of next parameter list

The Z80A-CTC list was organized in a similar manner.

The functions listed in Table XI established the need for four queues. Two of the queues (NWTXQ and LOTXQ) were shared with processor #1. Their use was discussed previously. The other queues which were local queues were designated the network receive queue (NWRXQ) and the network already transmitted queue (NATXQ). The NWRXQ was needed to store the memory block storage address of a correctly received network message pending further processing by the main operating system. The NATXQ was needed to store the memory block storage address of a transmitted network message pending receipt of an acknowledgement for that message. The initialization routine set the head and tail of the NWTXQ, NWRXQ and NATXQ to their respective start addresses.

One other task was completed during the initialization phase. This was to allocate a memory block storage address to the interrupt routine which received network messages. Since only a single communication channel was being tested, the alternative registers set was used to store the information needed to receive a network message. If additional links were added, this information would have to be stored in memory.

Operating System #2 Generalized Subroutines

The generalized subroutines for operating system #2 consisted of routines to add/delete information from the different queues and from the memory table. These routines were identical to those of operating system #1 except for the lock-out code. Since they have been discussed previously, they will not be repeated in this section.

Interrupt Service Routines
Operating System #2

The interrupt service routines required for operating system #2 were established by the operational characteristics of the Z80A-SIO. There were four different conditions for each port which caused a unique interrupt address to be generated. These conditions were port transmit buffer empty, external/status change, receive character available and special receive condition. The external/status change interrupt would be generated if the different control signals between the SIO and a modem changed. Since it was

not envisioned the universal network interface device would be tested using a modem, a routine was not developed for this interrupt condition.

Operating System #2 Transmit Routine. The approach used for the transmit routine was a deviation from the above. While the SIO did have the capability to generate an interrupt on a transmit buffer empty, it also had the capability to utilize a programmed I/O technique. Since for an interrupt situation the code would be almost identical to the 2651 case, a subroutine was developed using the programmed I/O capability. This provided another representative code which could easily be modified into an interrupt service routine. In addition, it allowed a faster transmission rate on the receive side as the alternative register set was used for receive storage information. The flowchart for the transmission subroutine is shown in Figure 5-14.

Operating System #2 Receive Routine. The interrupt service routine for receipt of network messages was simplified through the use of the alternative register set. Only eight instructions were required to accomplish reception. These instructions could be executed in 55 clock cycle which equated in the Z80A case to a transmission rate of approximately 575 kb/s. This same technique could be used for transmission allowing half duplex transmission/reception of about 500 kb/s.

Fig. 5-9--Continued

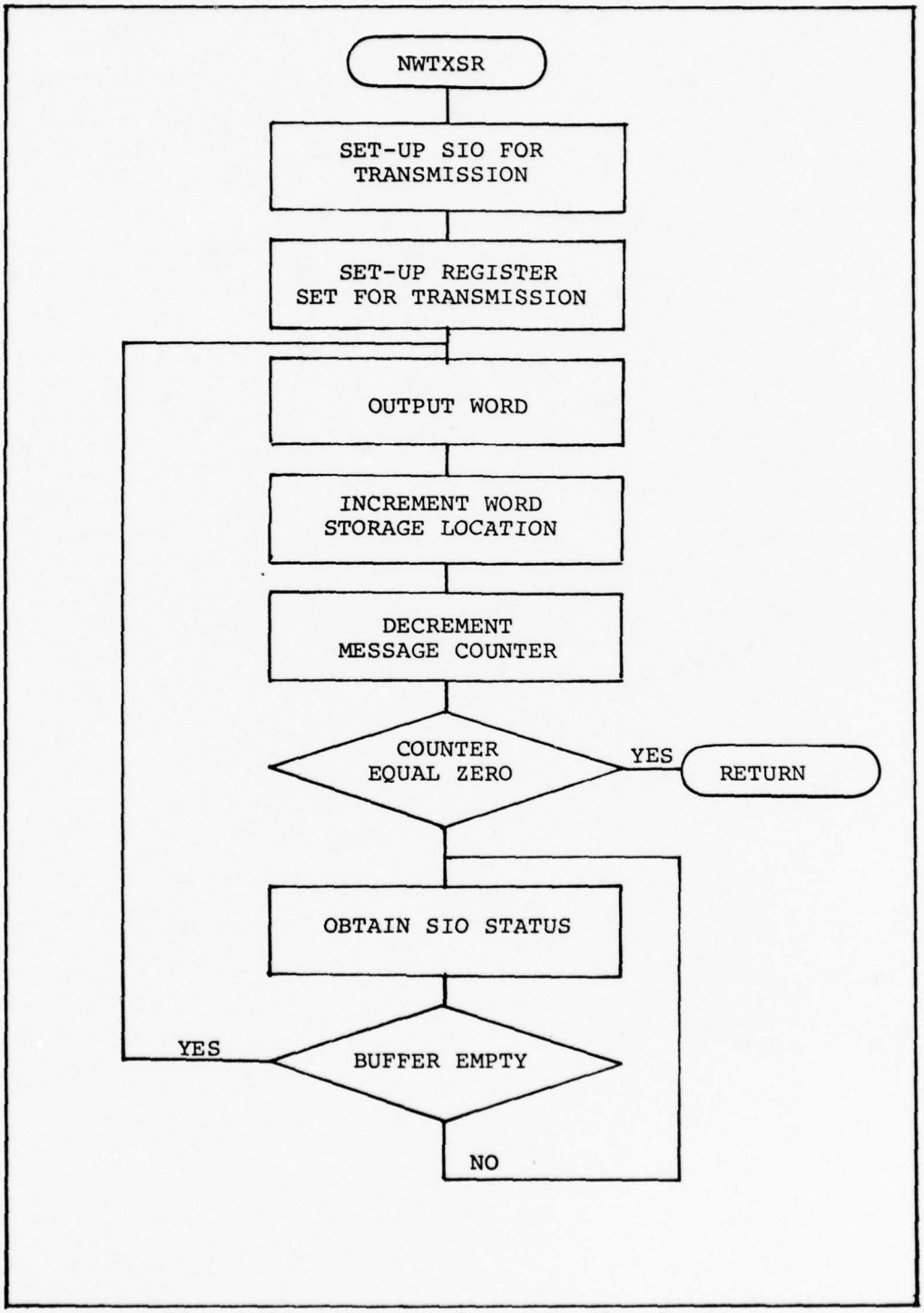


Fig. 5-14. Network Transmit Flowchart

Fig. 5-9--Continued

Operating System #2 Special Receive Interrupt Routine. A special receive condition interrupt was generated by a parity error condition, a RX overrun error condition, a CRC/framing error condition and an end of frame (SDLC) condition. The service routine had to differentiate between these conditions and then generate the appropriate action. The flowchart for the service routine is shown in Figure 5-15. For the case where the message was not error-free, the old block storage address was used to reinitialize the receive alternative register set.

Timer Interrupt Service Routine. One of the functions under service routine in Table XI still had not been developed. This function was the timer initialization associated with any message transmission. Negative acknowledgements are typically not employed in most link control protocols. Instead, an implied not receive correctly message is used. This is accomplished by using a timer to time out the amount of time after a message is sent until an acknowledgement for the message must be received. If the acknowledgement to the message is not received within that time frame, the message is automatically assumed to have been received incorrectly and is retransmitted. The timer interrupt service routine is shown in Figure 5-16.

Main Operating System #2

The main operating system for processor #2 performed a monitoring function. It monitored the NWTXQ and the

Fig. 5-10. Transmit Interrupt Service Routine Flowchart

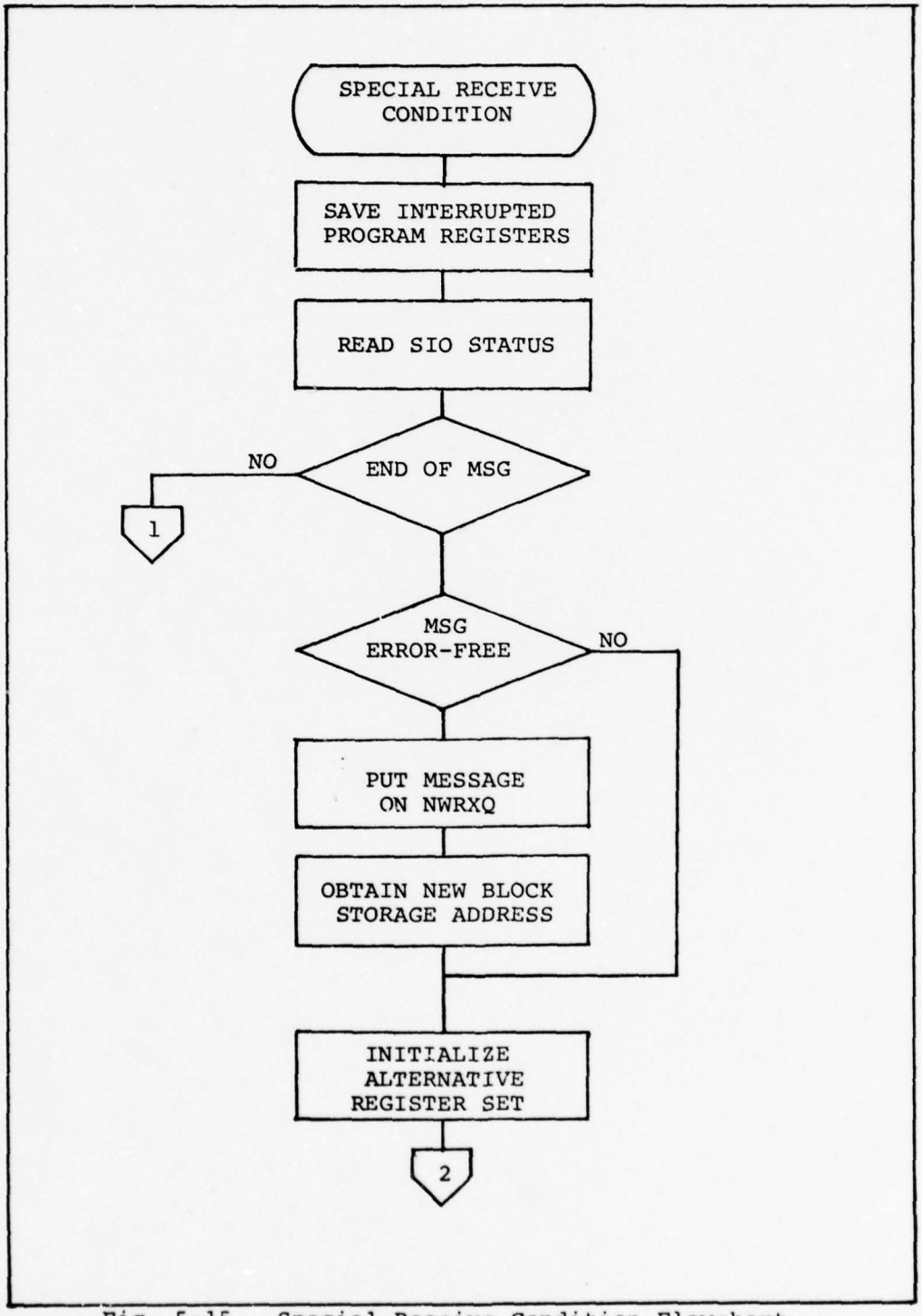


Fig. 5-15. Special Receive Condition Flowchart

Fig. 5-10--Continued

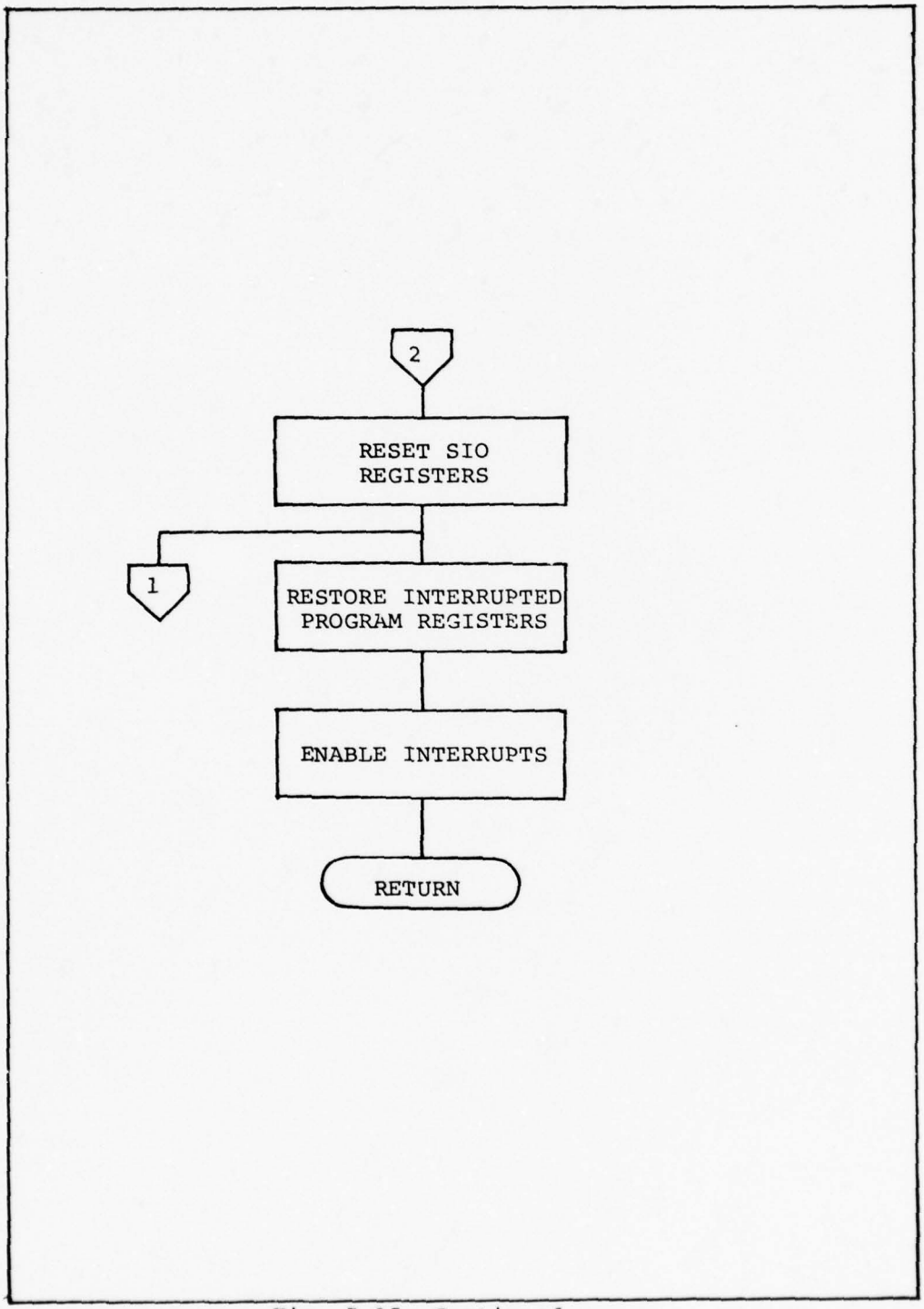


Fig. 5-15--Continued

would have been the only functions of the main operating

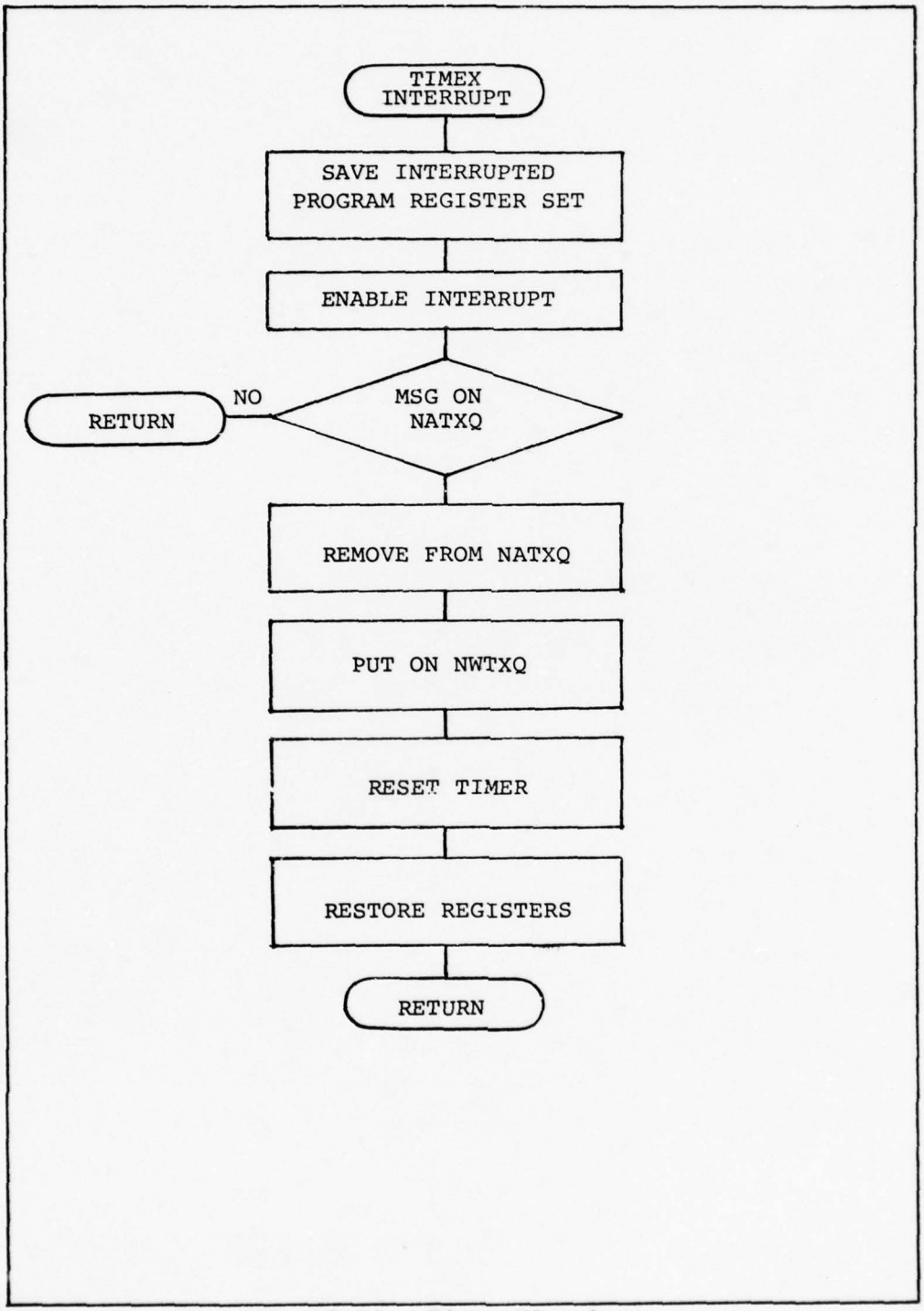


Fig. 5-16. Timer Flowchart

the main operating system. This segregation is accomplished in Table XI.

NWRXQ and performed the actions shown in Figure 5-17 and Figure 5-18. For testing purposes, it was assumed the acknowledgement function would be accomplished by a separate message whose address would be the address for the universal network interface device. A message addressed directly to the universal network interface device was assumed to be an acknowledgement control message.

The network address table was used to determine if the message was for a local subscriber connected to the universal network interface device. This table has been described previously.

Software Design Summary

This concluded the design of the software to support the universal network interface device. Certain functions shown in Table XI were not implemented. The processing of other control information would be link control protocol dependent and was left for user development. The determine-routing function was also not developed. This function increased code complexity without extending the concept being tested. To accomplish this function required another table lookup to determine what network port the message should be transmitted on. Once this was determined, the message could be transmitted or put onto a queue for that port.

The goal of the software design effort was to develop representative code which could be directly used in a real

Fig. 5-12. Local Transmit Queue Flowchart

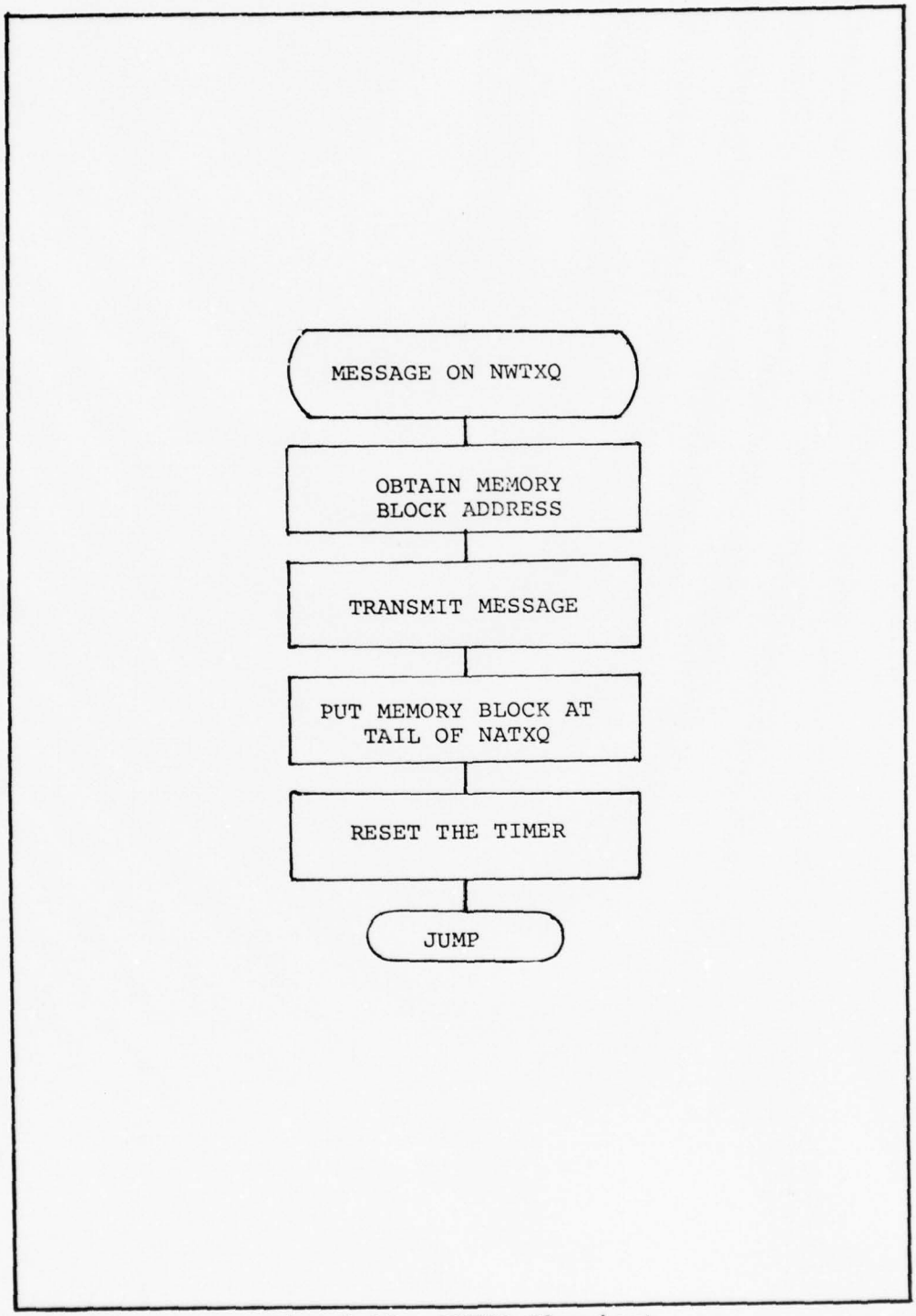


Fig. 5-17. NWTXQ Flowchart

Fig. 5-12--Continued

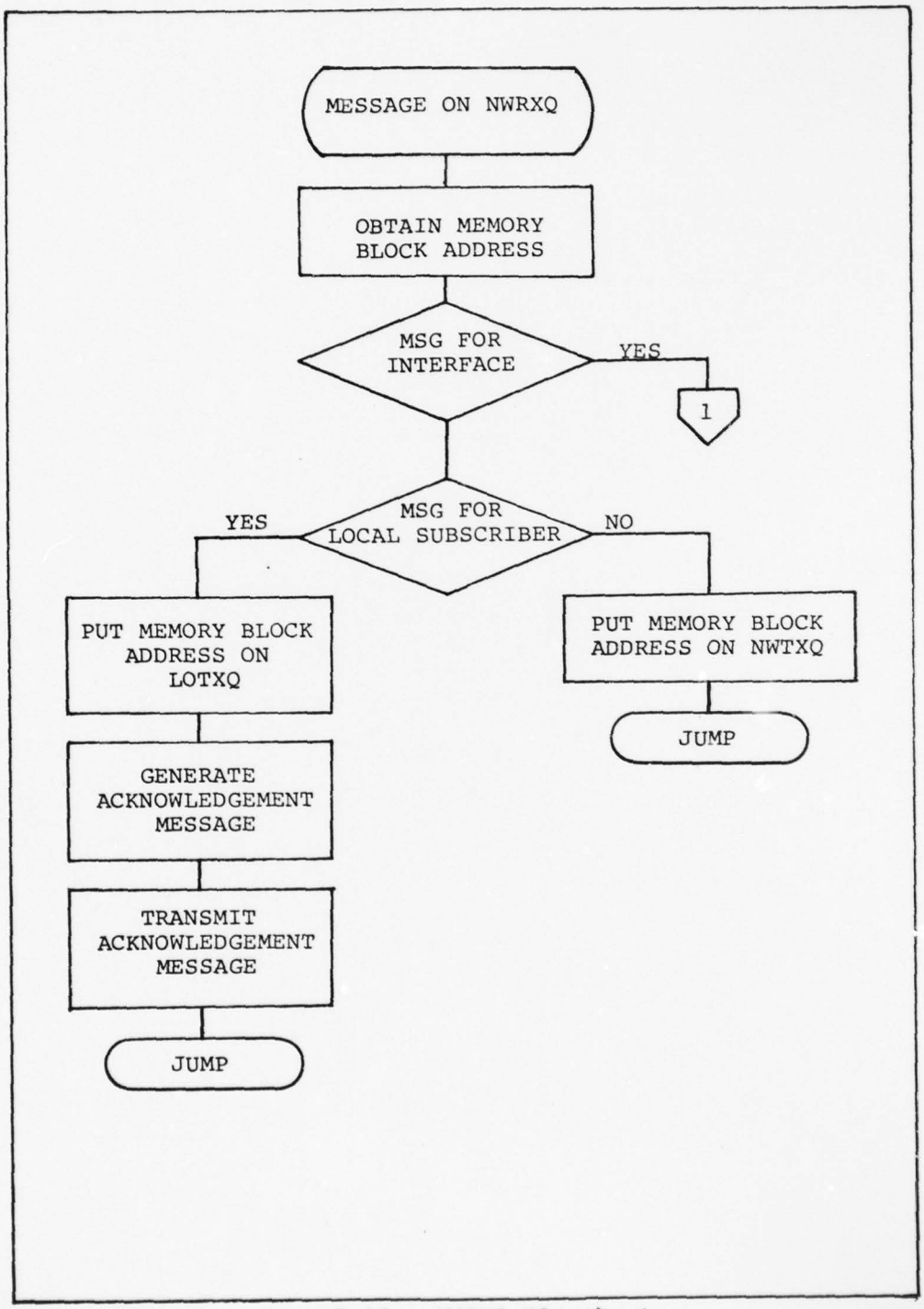


Fig. 5-18. NWRXQ Flowchart

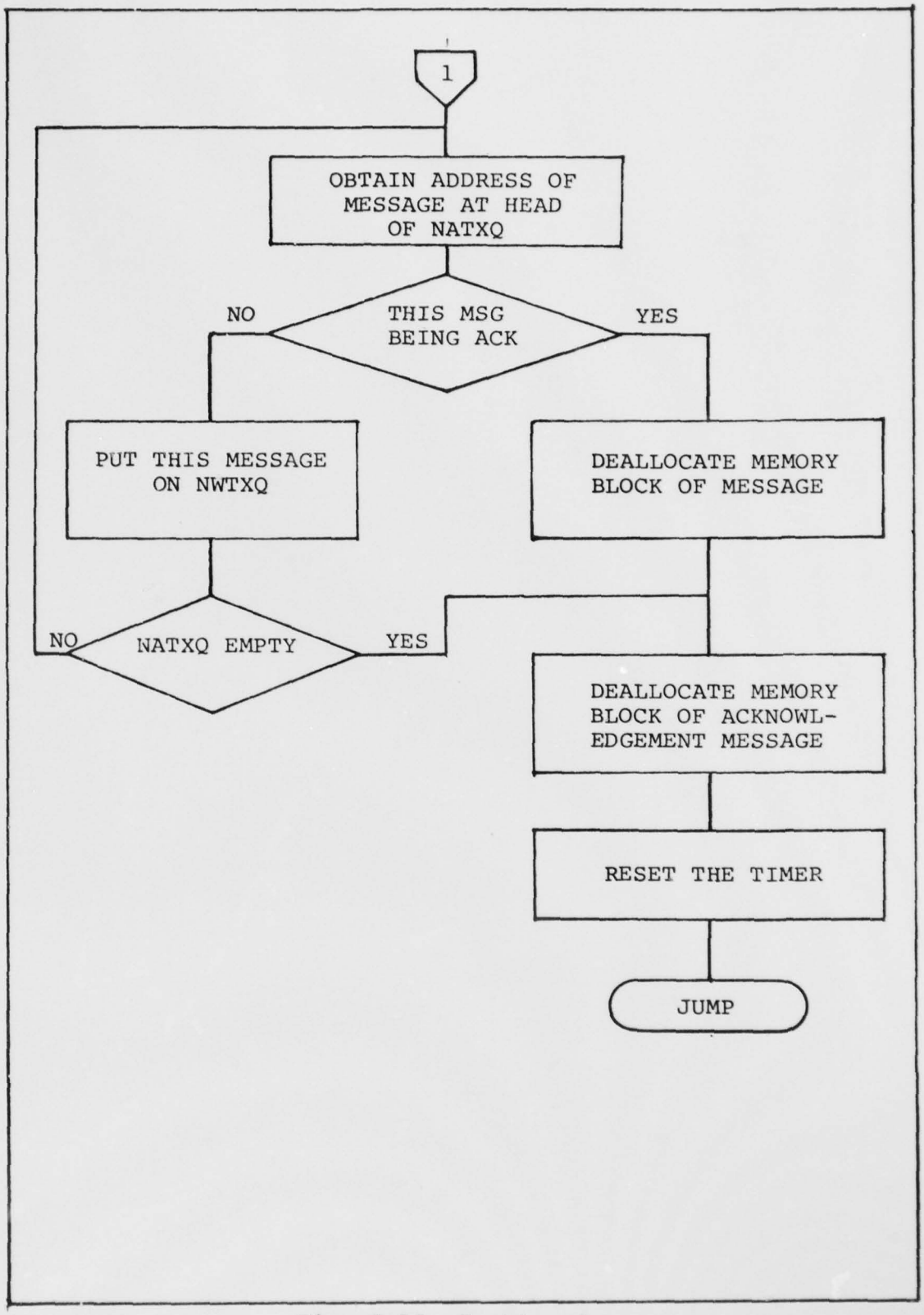


Fig. 5-18--Continued

network application and simplified code to allow testing of the universal network interface device. The code developed should allow the port A transmit section to be looped to the port A receive section. By using the pseudo link and local protocol developed, messages can be exchanged between terminals connected to the universal network interface device. This should allow the universal network interface design to be tested.

VI. Results and Recommendations

The primary objective of this thesis was to design and develop a small special purpose digital device which could be used for interfacing general peripheral devices to a communication network. The device was to be designed in such a manner as to be flexible enough to provide interfacing for a majority of peripherals into a majority of contemporary networks. The preliminary design for such a device has been completed and is discussed in the next section of the chapter. The final section presents recommendations for continuing the universal network interface device project.

Design Results

The design of the universal network interface device evolved into a modular design approach. This approach was used to provide a degree of universality to the device. The modular approach allows the user to configure the universal network interface device to his particular network environment.

To implement this modular concept, three cards were designed and the development of a fourth card was suggested. The first card, called the input card, provides RS-232C interfaces to connect modems or peripheral devices meeting the RS-232C standard to the universal network interface

device. Each input card has four individual full duplex ports with the number of ports expandable through use of additional input cards.

In a similar manner, a network card was designed to connect the universal network interface device into a communication network. Each network card consisted of two fully duplex RS 422/RS 423 ports with the number of ports expandable through use of additional network cards. The network card has the capability to be employed in networks having transmission speeds of up to 880 kb/s, although at this speed, half duplex, single-link communication could only be supported. To accommodate high speed, full duplex, multi-link operation required the basic network card be supplemented with a DMA capability. The design of such a card was left as a follow-on task to this investigation.

To provide a method to match the throughput of the universal network interface device to the network environment, a dual processor card was developed. This card allowed the universal network interface device to be upgraded to a two microprocessor (Z80A) configuration.

The software developed for the universal network interface device was structured to allow testing of the completed design. It was envisioned the dual process configuration would be tested and thus two operating systems were developed. The software developed did implement the packeting concepts for information transmission. In this configuration, the message is broken into a number of

submessages by the universal interface device and transmitted separately. The submessages are reassembled back into the complete message at the destination.

Recommendations

The recommendations for the universal network interface device involve the construction of an actual device. The first recommendation concerns the microprocessor board to be used. The design of the different cards was based upon the Z80-MCB although the processor selected was the Z80A. The proposed design must be reviewed upon release of the Z80A-MCB specifications to determine if the new board requires any modifications be made to the proposed design.

Once the validity of the design is verified, the following actions need to be accomplished:

1. Design the layout for the individual cards.

2. Construct the different cards. In this effort it is suggested the Z80-WWB be used. The Z80-WWB is compatible with the other boards in the Z80 series.

3. Test the software/hardware design. The software developed should allow messages to be interchanged between peripherals connected to the input card provided the network card port A TX output is connected to the port A RX input. The software developed can be assembled by a Mostek Z80 cross-assembler and loaded through the use of a Z80 PROM monitor.

4. Design and test a network card with a DMA capability. The design of this card should be such as to allow it to operate in a dual processor configuration.

Upon successful completion of the above, the performance of the universal network interface device needs to be evaluated. A determination of the throughput capability of the single and dual processor configuration is required to establish a throughput limitation for the device. These limitations should also be determined for the number of network and input ports which can be connected to the universal network interface device.

Bibliography

1. 1842 EEG/EEIC TR 78-5. An Engineering Assessment Toward Economic, Feasible and Responsive Base-Level Communications through the 1980's. Richard-Gebaur AFB, Missouri: 1842 Electrical Engineering Group, 31 October 1977.

2. Kleinrock, Leonard. Queueing Systems Volume II: Computer Applications. New York, New York: John Wiley and Sons, 1976.

3. ASD-TR-36-11. Management Guide to Avionic Software Acquisition, Volume I--An Overview of Software Development and Management. Dayton, Ohio: Logicon, 1976.

4. Myers, Ware. "The Need for Software Engineering," Computer, 11:12-24 (February 1978).

5. 9022-73.2. Structured Analysis Reader Guide. Waltham, Massachusetts: SofTech Inc., May 1975.

6. 9022-78R. An Introduction to SADT, Structured Analysis and Design Technique. Waltham, Massachusetts: SofTech, Inc., November 1976.

7. Manaly, John R. Design of a Laboratory Data Acquisition System (Time Digitization System). MS thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, March 1978.

8. Martin, James. System Analysis for Data Transmission. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1972.

9. Report Number 1822. Specifications for the Interconnection of a Host and an IMP. Advanced Research Project Agency. Cambridge Massachusetts, Bolt Beranek and Hewman, Inc., April 1973. (AD 759 433).

10. Weissberger, Alan J. Data Communication Handbook. Sunnyvale, California: Signetics, October 1977.

11. Datapro. The EDP Buyer Bible. Delran, New Jersey: Datapro Research Corporation, November 1978.

12. ETA Standard RS-232C. Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange. Washington, D.C.: Electronic Industrial Association, August 1969.

13. MIL-STD-188-100. Common Long Haul and Tactical Communication System Technical Standards. Washington, D.C.: Department of Defense, 15 November 1972.

14. MIL-STD-188-114. Electrical Characteristics of Digital Interface Circuits. Washington, D.C.: Department of Defense, 24 March 1976.

15. Morris, Dusty. "Resived Data-Interface Standards," Electronic Design, 18:138-141 (1 September 1977).

16. Greene, William and Udo W. Pooch. "A Review of Classification Schemes for Computer Communication Networks," Computers, 10:12-20 (November 1977).

17. Schwartz, Mischa. Computer-Communication Network Design and Analysis. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1977.

18. Doll, Dixon R. Data Communications Facilities, Networks and Systems Design. New York: New York: John Wiley and Sons, 1978.

19. 310-D70-30. DCS AUTODIN Switching Center and Tributary Operations. Washington, D.C.: Defense Communication Agency, June 1970.

20. Osborne, Adam. An Introduction to Microcomputers Volume II Some Real Products. Berkeley, California: Adam Osborne and Associates, Incorporated, June 1977.

21. Zilog, Inc. Z80 Assembly Language Programming Manual. Cupertino, California: Zilog, Inc., January 1978.

22. Zilog, Inc. Z80-MCB Hardware User's Manual. Cupertino, California: Zilog, Inc., January 1978.

23. Klingen, Edwin E. Microprocessor System Design. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1977.

24. Zilog, Inc. The Z80 Family Program Interrupt Structure. Cupertino, California: Zilog, Inc., October 1977.

25. Intel, Corp. MCS-80 User's Manual. Santa Clara, California: Intel Corporation, October, 1977.

26. Texas Instrument, Inc. The TIL Data Book (Second Edition). Dallas, Texas: Texas Instruments, Inc., 1976.

27. Zilog, Inc. Product Specification Z80-CPU/Z80A-CPU. Cupertino, California: Zilog, Inc., March 1978.

28. Zilog, Inc. Product Specification Z80-SIO. Cupertino, California: Zilog, Inc., March 1978.

29. EIA Standard RS-423. Electrical Characteristics of Unbalanced Voltage Digital Interface Circuits. Washington, D.C.: Electronic Industries Association, April 1975.

30. EIA Standard RS-422. Electrical Characteristics of Balanced Voltage Digital Interface Circuits. Washington, D.C.: Electronic Industries Association, April 1975.

31. Motorola Semiconductors Products, Inc. Master Selection Guide and Catalog. Phoenix, Arizona: Motorola, Inc., 1977.

32. Fairchild Semiconductor. Linear Integrated Circuits Data Book. Mountain View, California: Fairchild Camera and Instrumental Corporation, 1976.

33. RADC-TR-74-258. Air Force Communications Service Digital Transmission Study Volume II Digital Cable System Handbook. Griffiss AFB, New York: Rome Air Development Center, September 1974. (AD A000 022)

34. Loewer, Bob. "The Z-80 In Parallel," Byte, 3:60-63, 174-176 (July 1978).

35. IBM Corp. IBM Synchronous Data Link Control General Information. Research Triangle Park, North Carolina: International Business Machines Corporation, May 1975. (GA27-3093-1)

36. Stone, Harold S. Introduction to Computer Architecture. Chicago, Illinois: Science Research Associates, Inc., 1975.

Appendix A

Structured Analysis Diagrams (Ref 7)

This appendix gives a short description of how Structured Analysis models are constructed and explains the SA diagram conventions used in this paper. It must be noted that the format used to present the models in this paper is not standard according to the rules developed by SofTech. The changes were made to present the models in a manner which is more familiar to readers who have no experience with SA models. Although the format is not that used by SofTech, the diagrams of the models are organized and related according to SofTech procedures, and the conventions used to construct individual diagrams are standard.

The Structured Analysis Design Technique is a general purpose top-down modular technique for modeling functions. The functions may be as varied as farming or manufacturing, but SA was developed primarily as a software requirements definition and design tool. Although a complete SA model actually consists of two models, one for activities and one for data, this paper employs only activity models so the conventions described here are those which apply to activity models.

An SA activity model consists of a series of diagrams which present in progressively more detail the activities

necessary to perform some function. Each diagram represents a self-contained activity which is part of the overall function. A diagram shows how its activity is decomposed into subactivities, and how the subactivities are related to each other. The subactivities in each diagram may then be decomposed on separate diagrams which leads to a tree structure of several levels. At the top is one diagram which represents the whole function, and at the bottom are the diagrams which show the most detailed activities.

Figure A-1 shows how an SA model would appear if all the diagrams were on one page. Of course, in real SA diagrams only one level of decomposition is shown, but the figure demonstrates the top-down nature of SA and the way activities are grouped into modules. In the figure, as in real models, one large box represents the whole function, and that is decomposed into successive levels of related activities. The decomposition process continues until the desired amount of detail has been developed, which may require more levels than shown in Figure A-1. Another thing to note is that while the figure shows only 3 subactivities in each decomposition, any number from 3 to 6 is acceptable.

From Figure A-1, it should be apparent that SA diagrams are constructed with boxes and arrows. In an activity model, each box represents an activity, and is called a node. Arrows represent "data" where the word data is used in a very general sense to include anything that is not an activity. Figure A-2 shows the different meanings

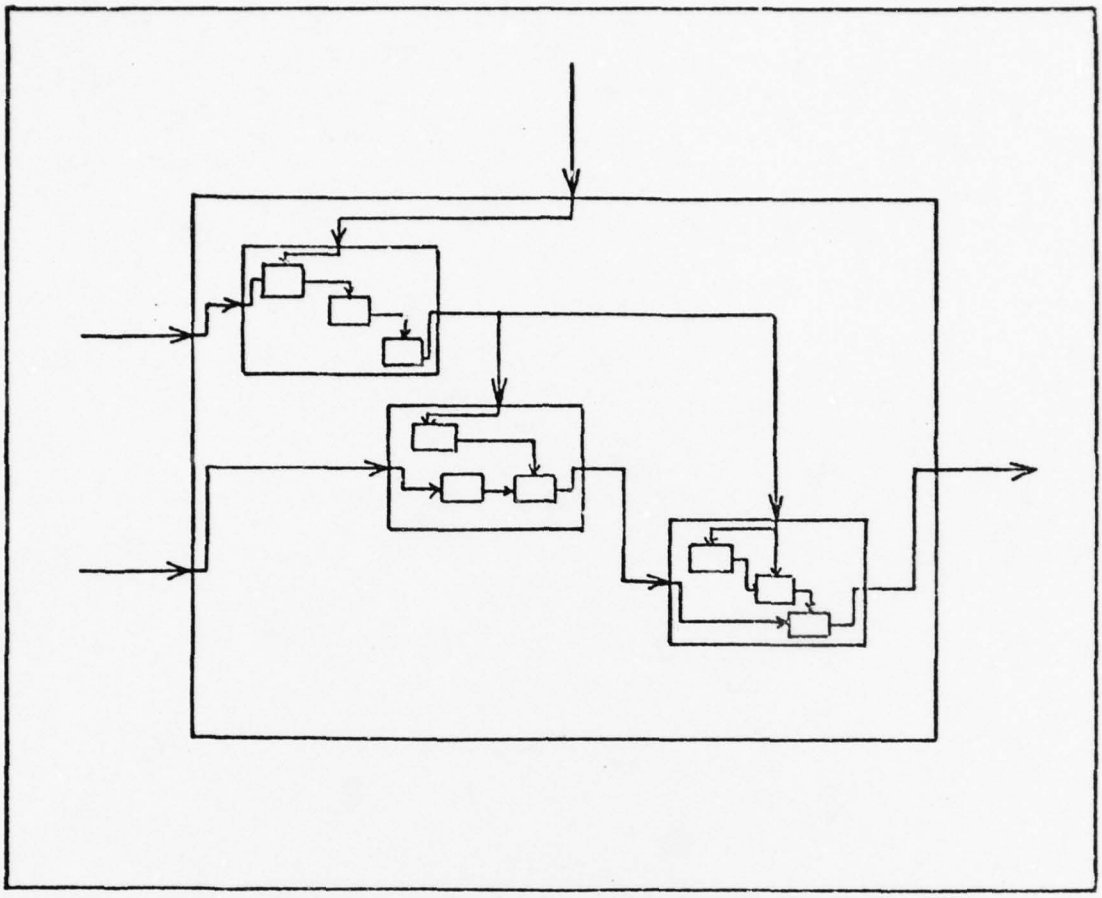


Fig. A-1. Top-down View of an SA Model (Ref 7:162)

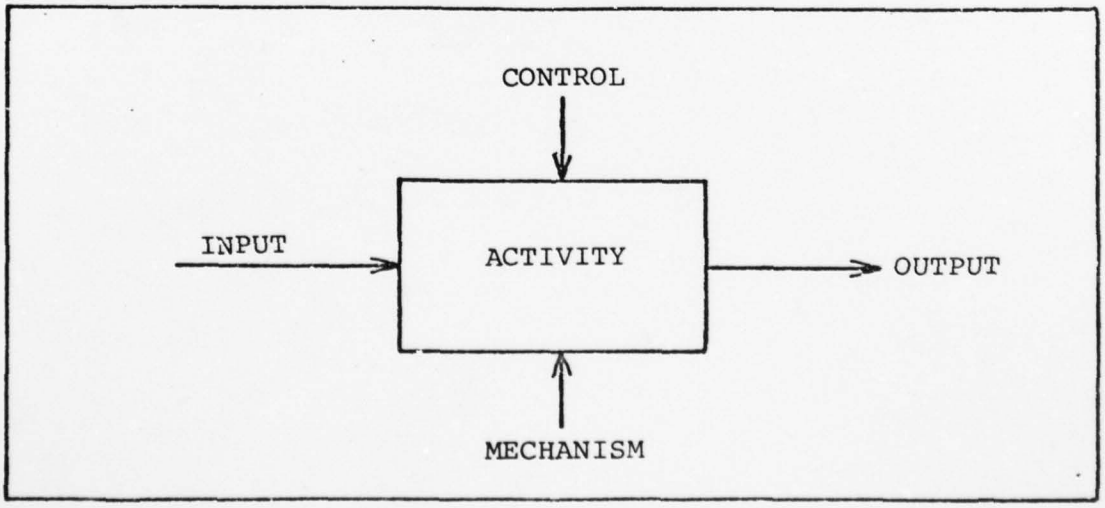


Fig. A-2. Arrow Definitions (Ref 7:162)

given to arrows depending on which side of a box they enter or leave. An input is data that is modified by the activity to produce an output. A control is data which may or may not be converted into output, but which in some way restricts the activity (starts or stops it for example). A mechanism is a person or thing which acts as a processor. Mechanism arrows are often omitted when the processor is the same for all nodes. No limit is placed on the number of arrows which may interface with a side of a box, but it is common practice to group related types of data.

Between boxes, arrows may split and join. In general, all branches of an arrow contain the same data unless a branch is given a separate label. This convention is summarized in Figure A-3 which also gives two forms of OR-branches. The OR-branches are used to show that data follows one path or the other, but not both.

When two nodes are related so that the output of each is a control for the other, a special two-way arrow may be used. Figure A-4 shows a mutual control situation with a two-way arrow and the equivalent form with normal arrows. An arrow showing mutual control has two labels separated by a slash; the first label identified data going forward, and the second is the feedback data.

A special numbering system is used to distinguish between nodes at different levels and between nodes at the same level. In an activity model, node numbers are prefixed with the letter A. For preliminary nodes, A is

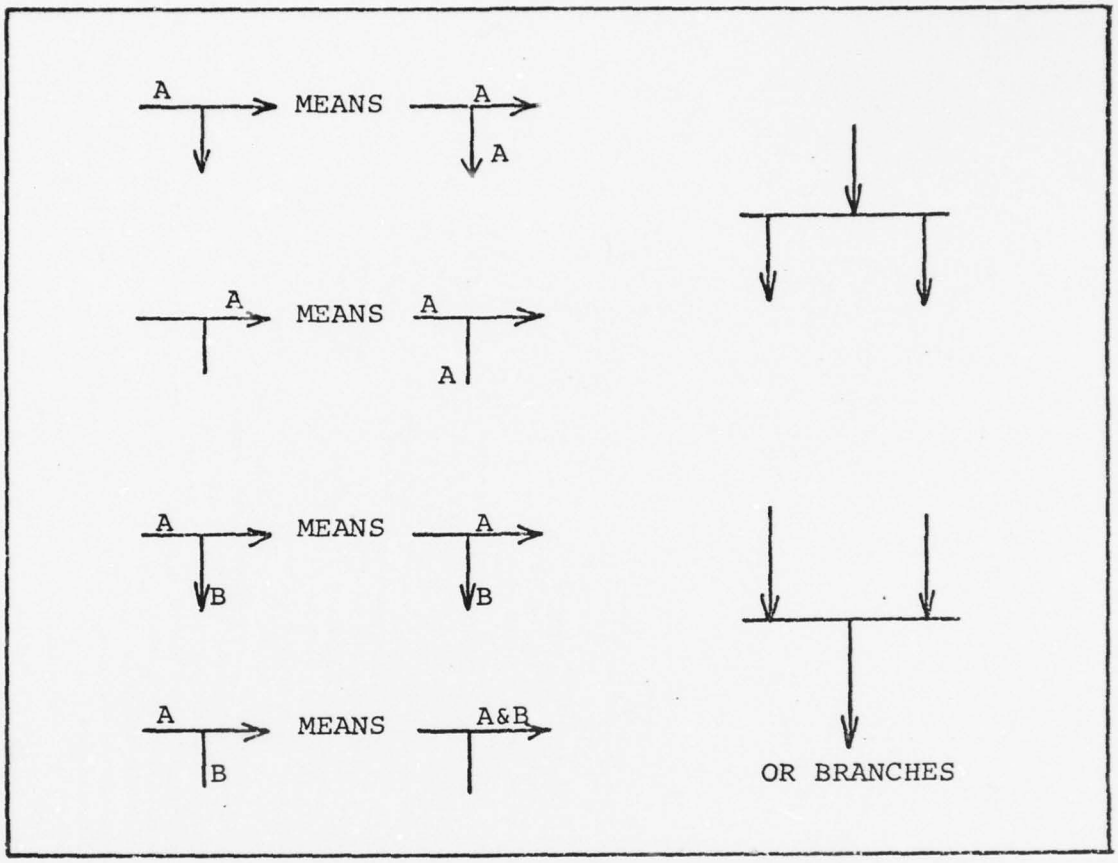


Fig. A-3. Arrow Branches (Ref 7:164)

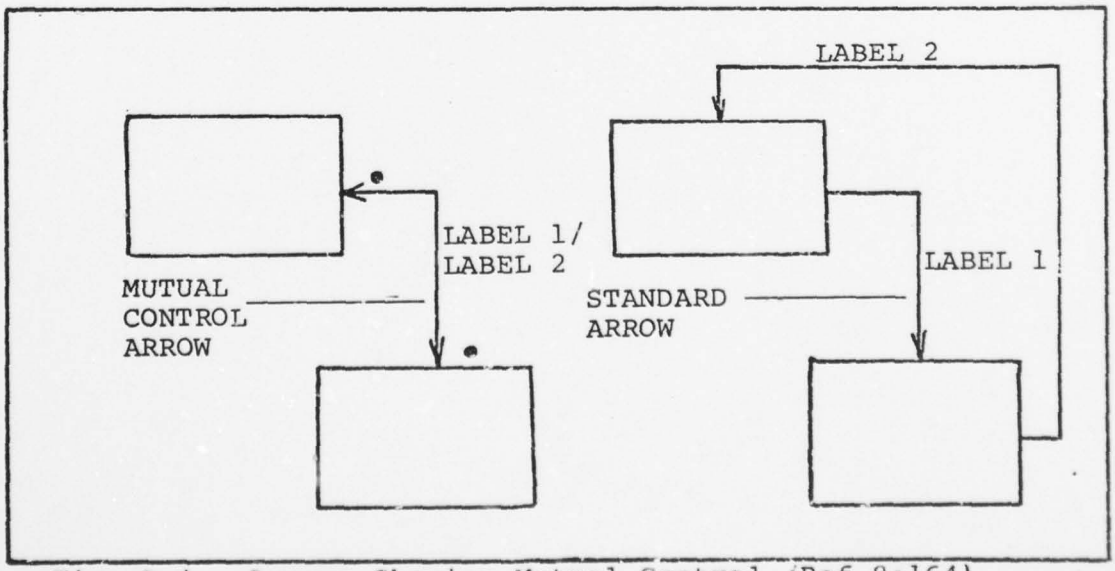
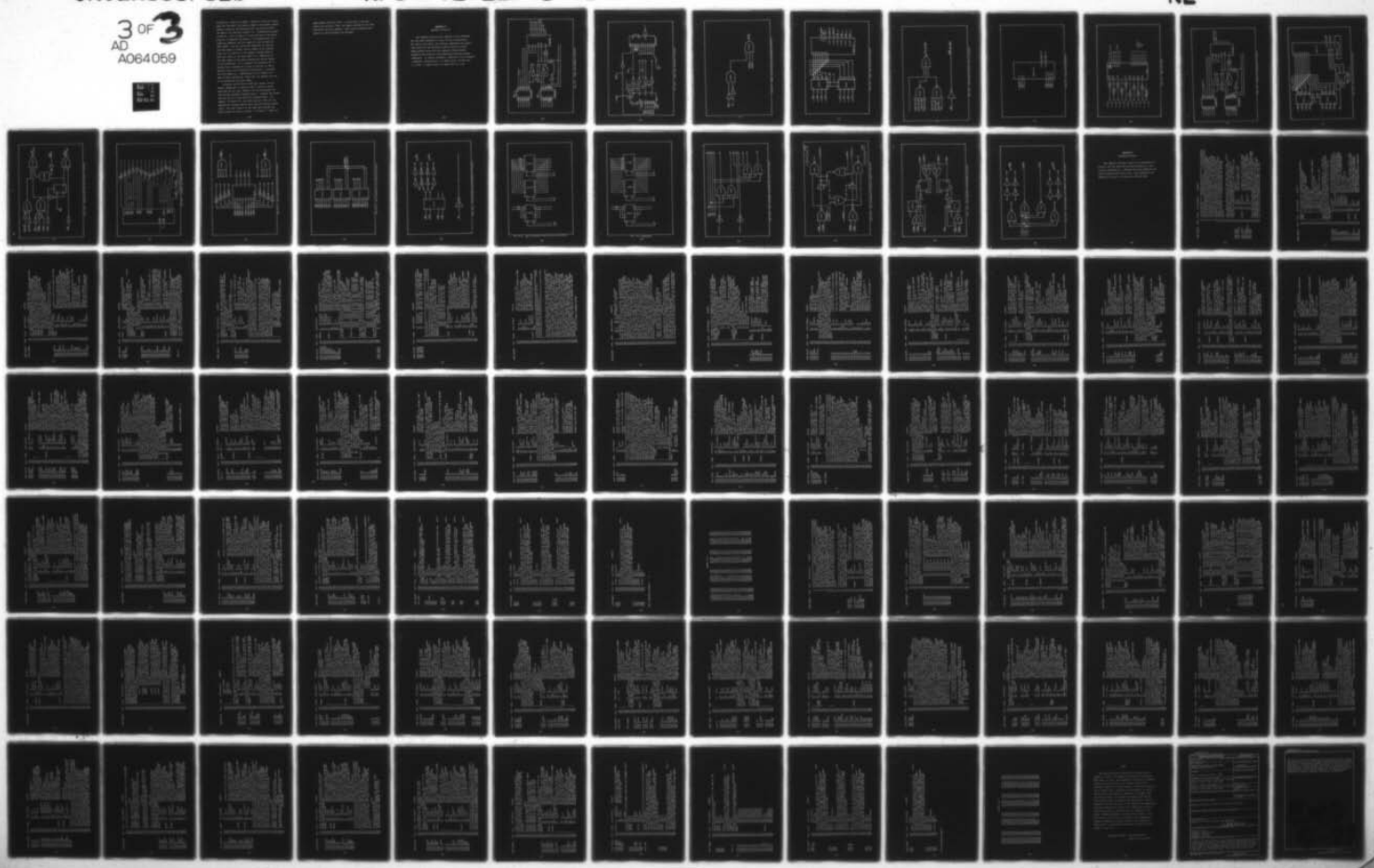


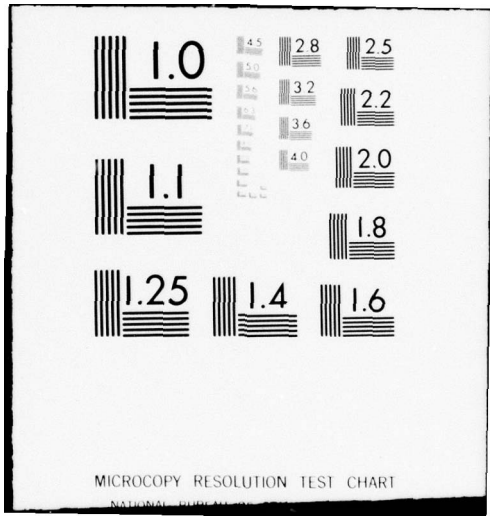
Fig. A-4. Arrows Showing Mutual Control (Ref 8:164)

AD-A064 059 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
PRELIMINARY DESIGN OF A UNIVERSAL NETWORK INTERFACE DEVICE.(U)
DEC 78 S C SLUZEVICH

UNCLASSIFIED AFIT/GE/EE/78-41 NL

3 OF 3
AD
A064059




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART

followed by a dash and a number. Node A-0 serves a sheet for the model; the node is simply a box showi outputs, controls, and mechanisms for the function the model is to describe (Figure 2-2). Decompositi in Node A0. Note in Figure 2-3 that each box of th position is numbered; the boxes on all decompositic grams are numbered, and this number is used to form node number. For the activities subordinate to Nod the node number is simply the box number on A0; Pro Local Info in Figure 2-3, for example, becomes Node From this level on, the node number is a combinatic the node number of the parent diagram and the box n of the subordinate. As an example, the decompositi Process Local Information is given in Figure 2-4, F Receive Local to be Transmitted Information, is ass the node number A11. Subordinates of A11 (Figure 2 such as Store Information, would have the numbers A so on through the last box number.

A special code called an ICOM code (Input, Co Output, Mechanism) is normally used to identify ar This code was not used, however, for the SA diagra the Universal Network Interface Device. Instead, into and out of a given node were assigned names. example, in Figure 2-5, the arrows going to node A labeled next storage address (A12) and next storag (A12). The (A12) portion of the name thus provide node to which the arrow is going. In Figure 2-7

names appear suffix by (All). In this case, since the arrows are entering a node, the number provided is the node from which the arrow emerged. This to/from numbering convention is used throughout the SA model.

Appendix B

Hardware Circuitry

This appendix provides the complete circuit diagrams for the cards designed as a part of this investigation. For each of the cards, the different components are identified along with the different signals within the card. These signals along with the component identification establish the different interconnections required between components. In certain instances, a NOT gate is not assigned a component identification. In those cases, the NOT gate is a 7404. A jumper option is identified by a —O.

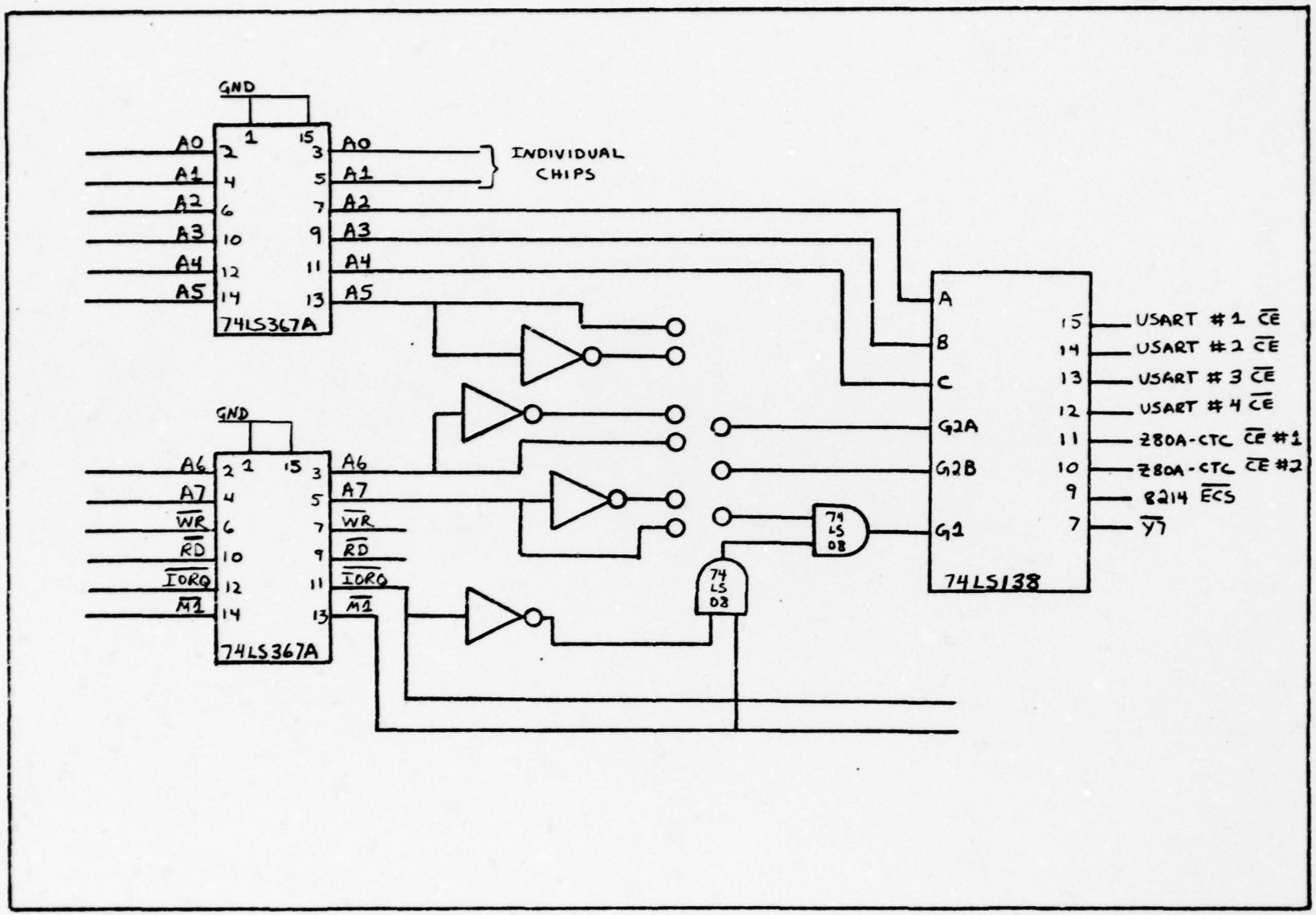


Fig. B-1. Input Card Address Circuitry

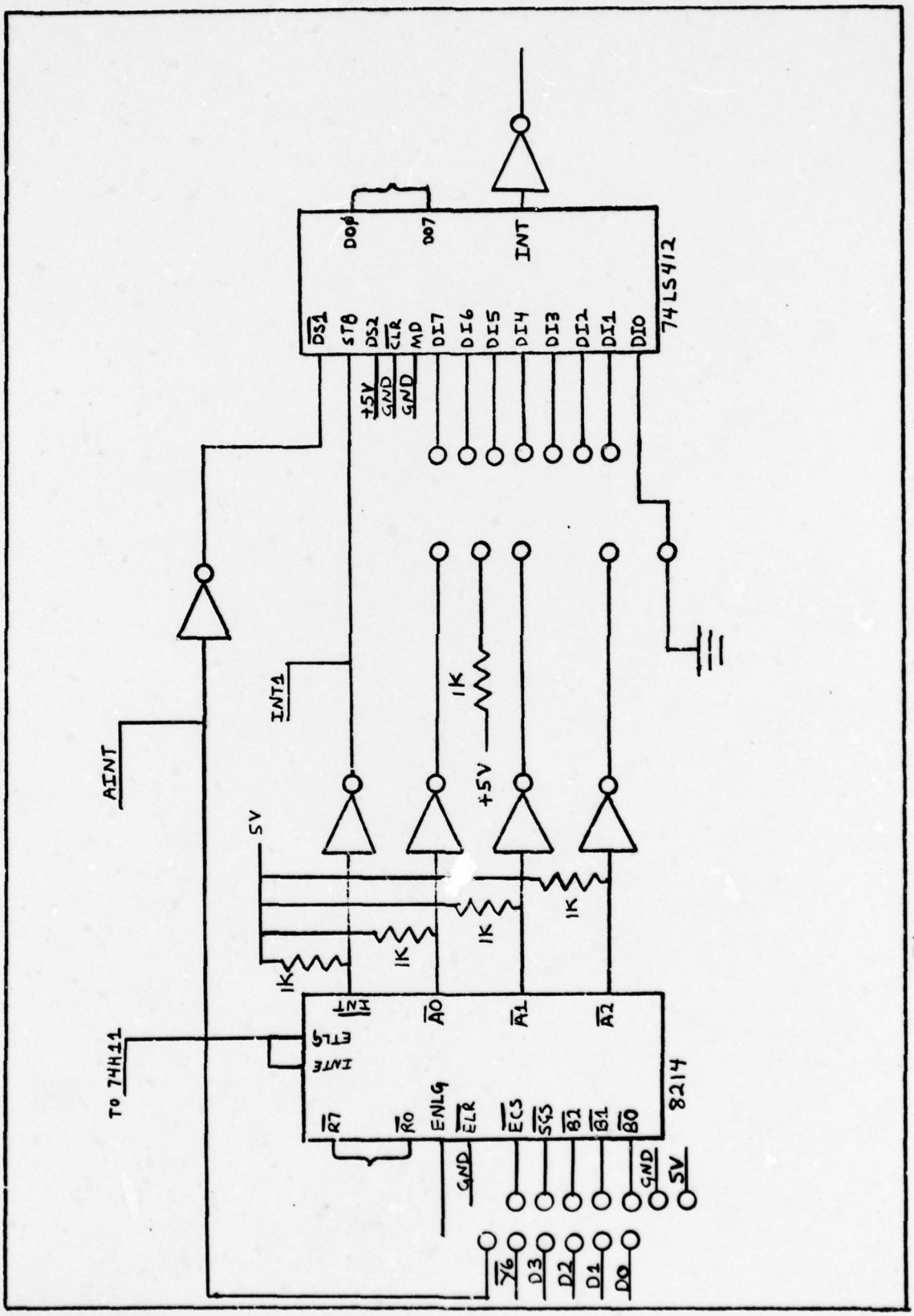


Fig. B-2. Input Card Interrupt Circuitry

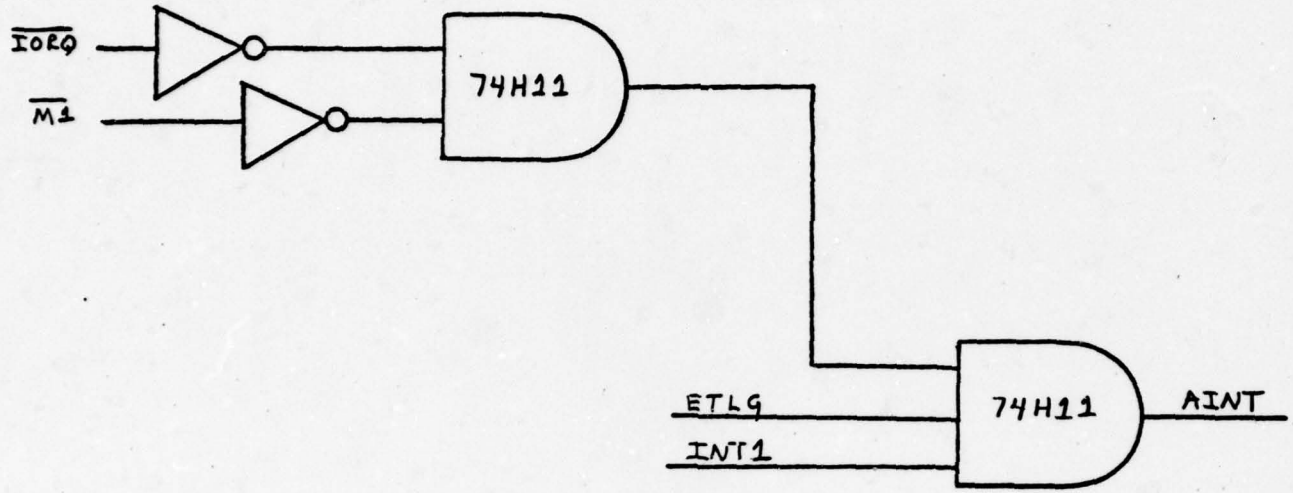


Fig. B-3. Input Card Interrupt Acknowledge Circuitry

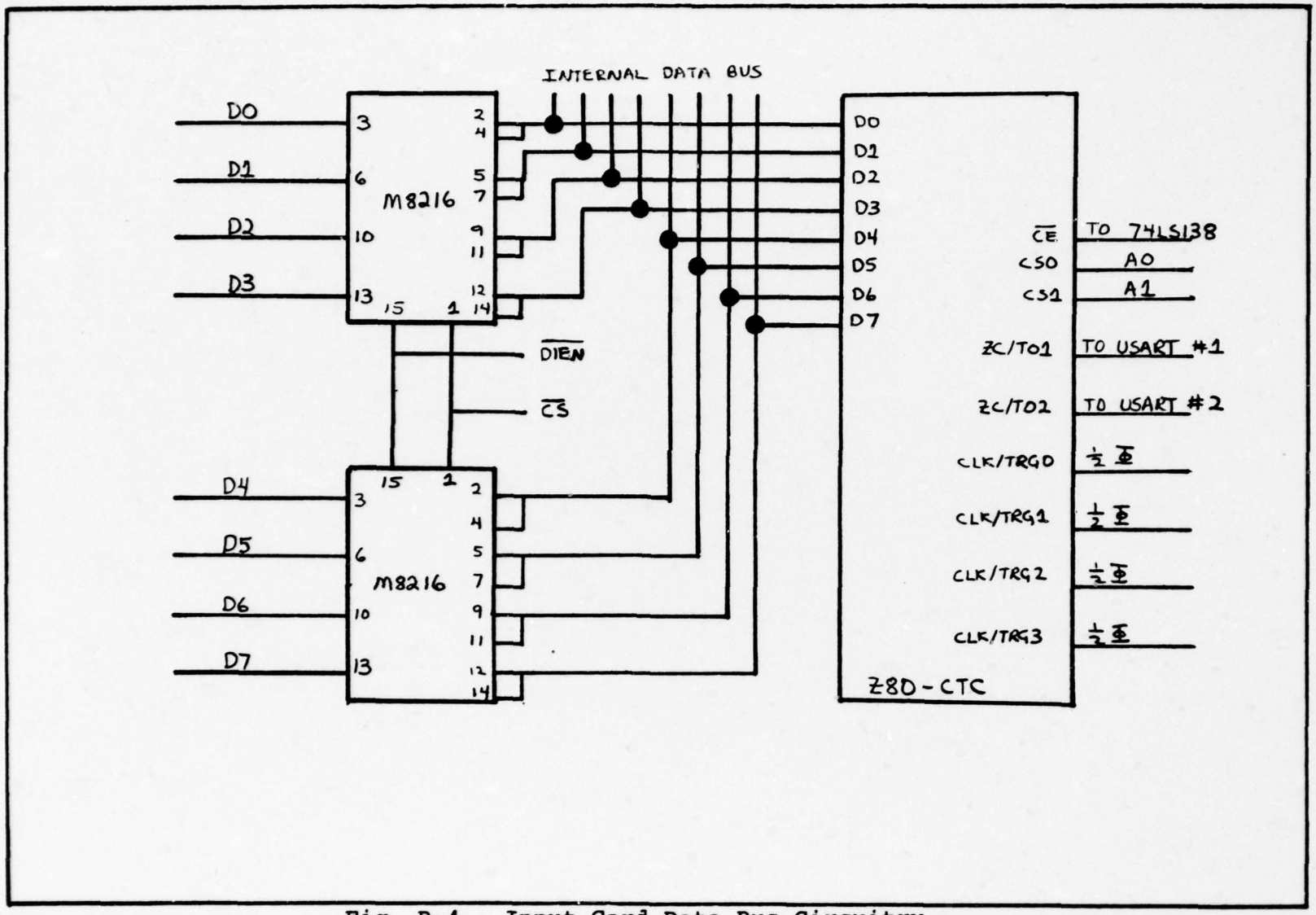


Fig. B-4. Input Card Data Bus Circuitry

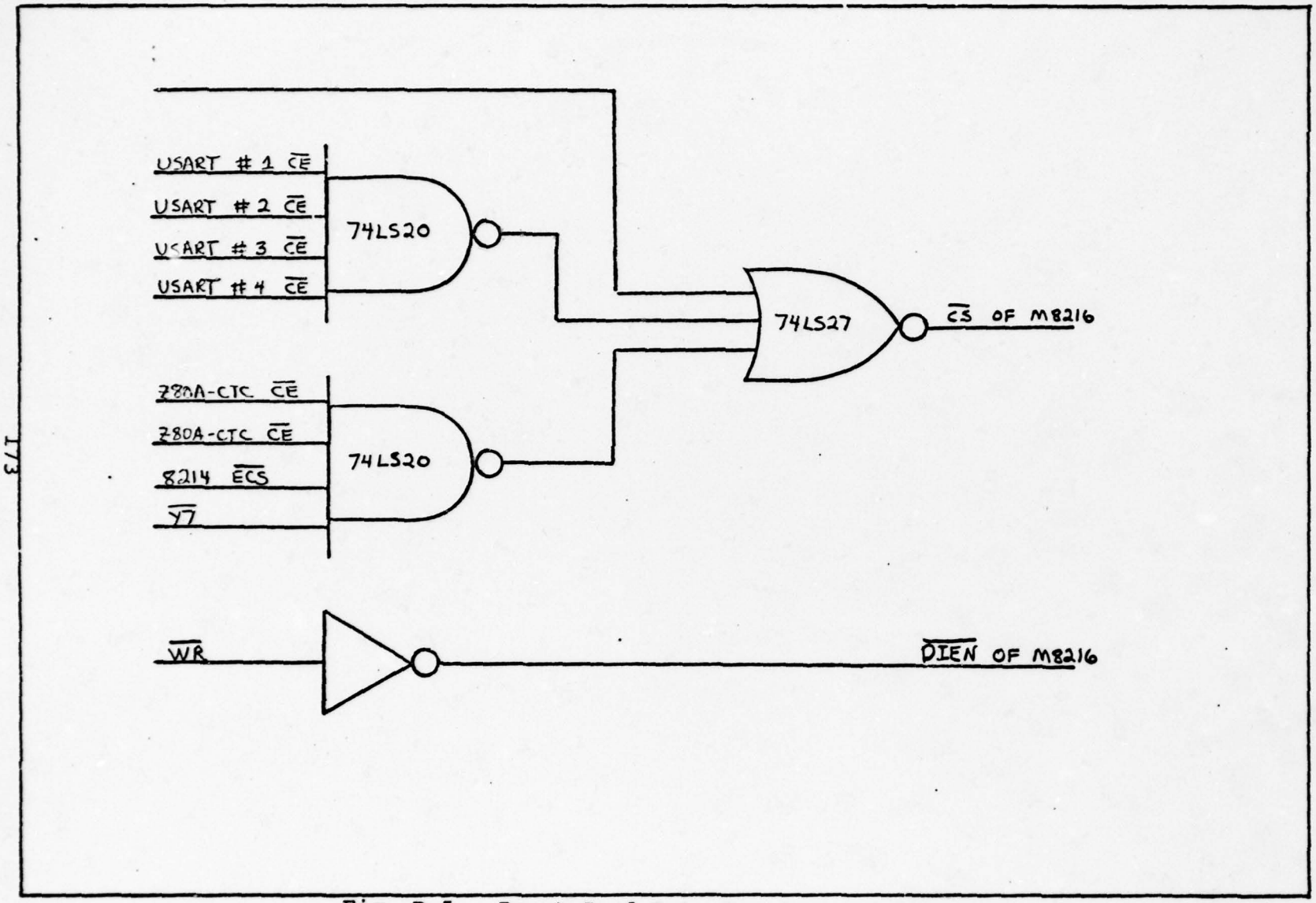


Fig. B-5. Input Card Data Bus Control Circuitry

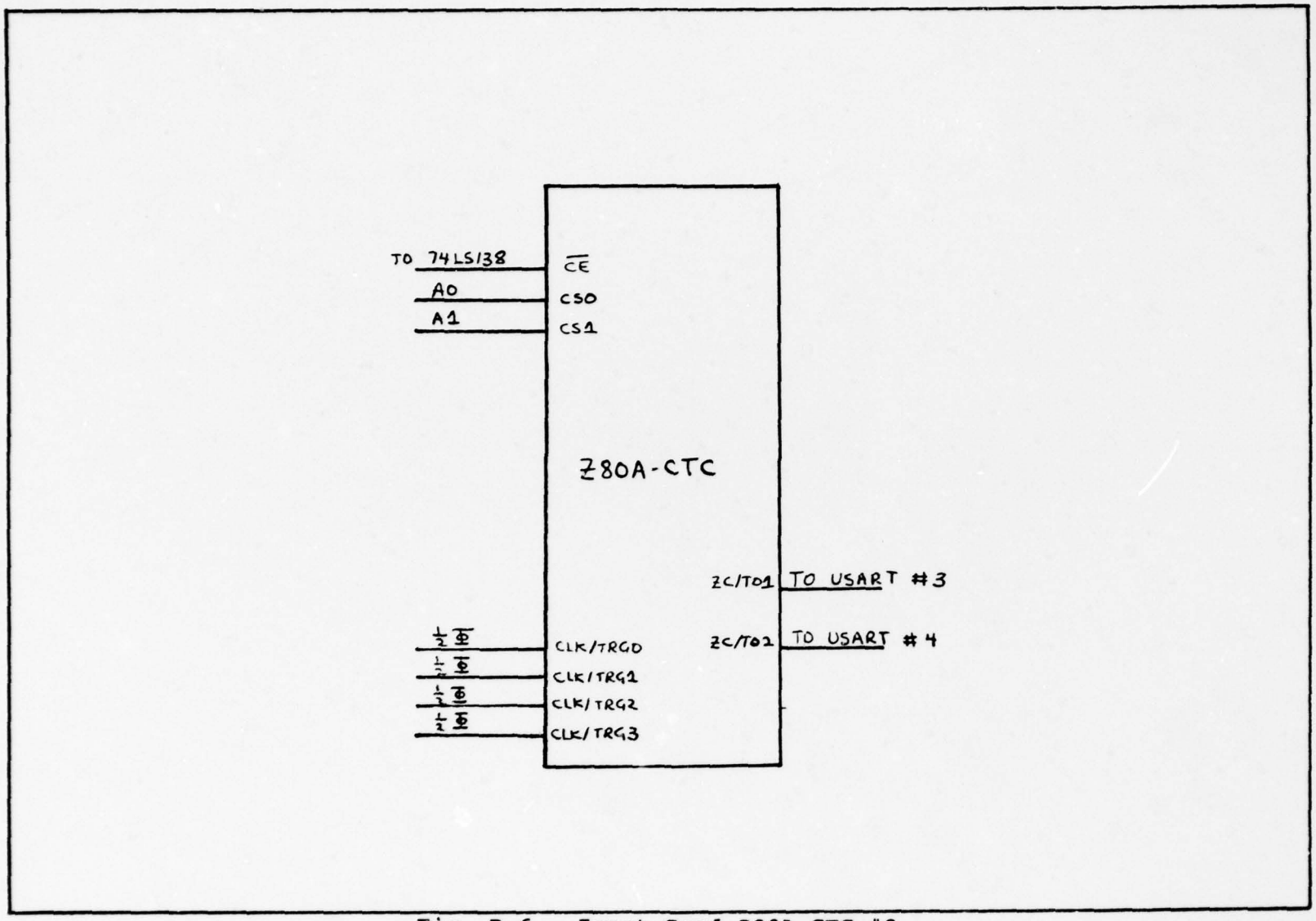


Fig. B-6. Input Card Z80A-CTC #2

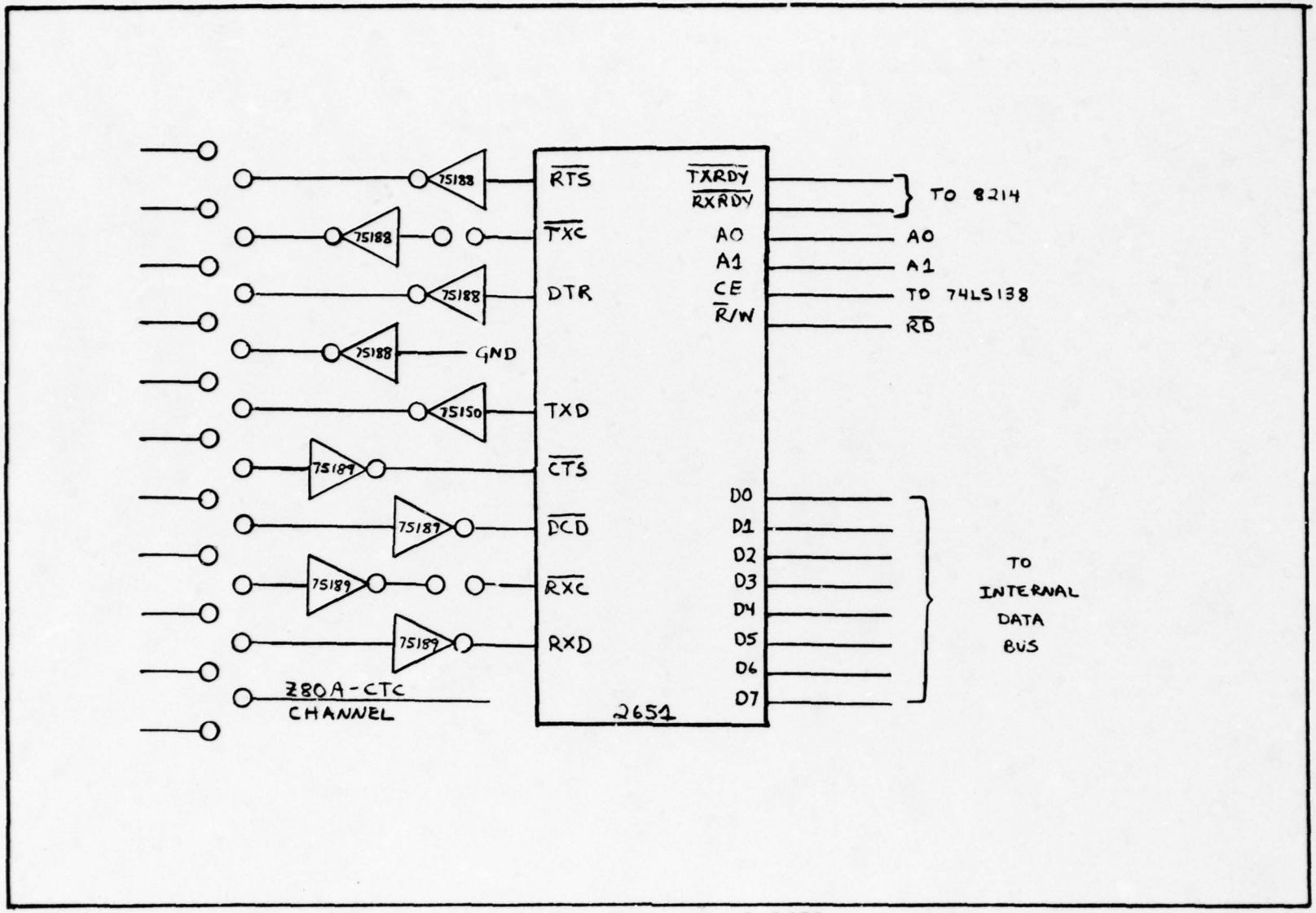


Fig. B-7. Input Card 2651

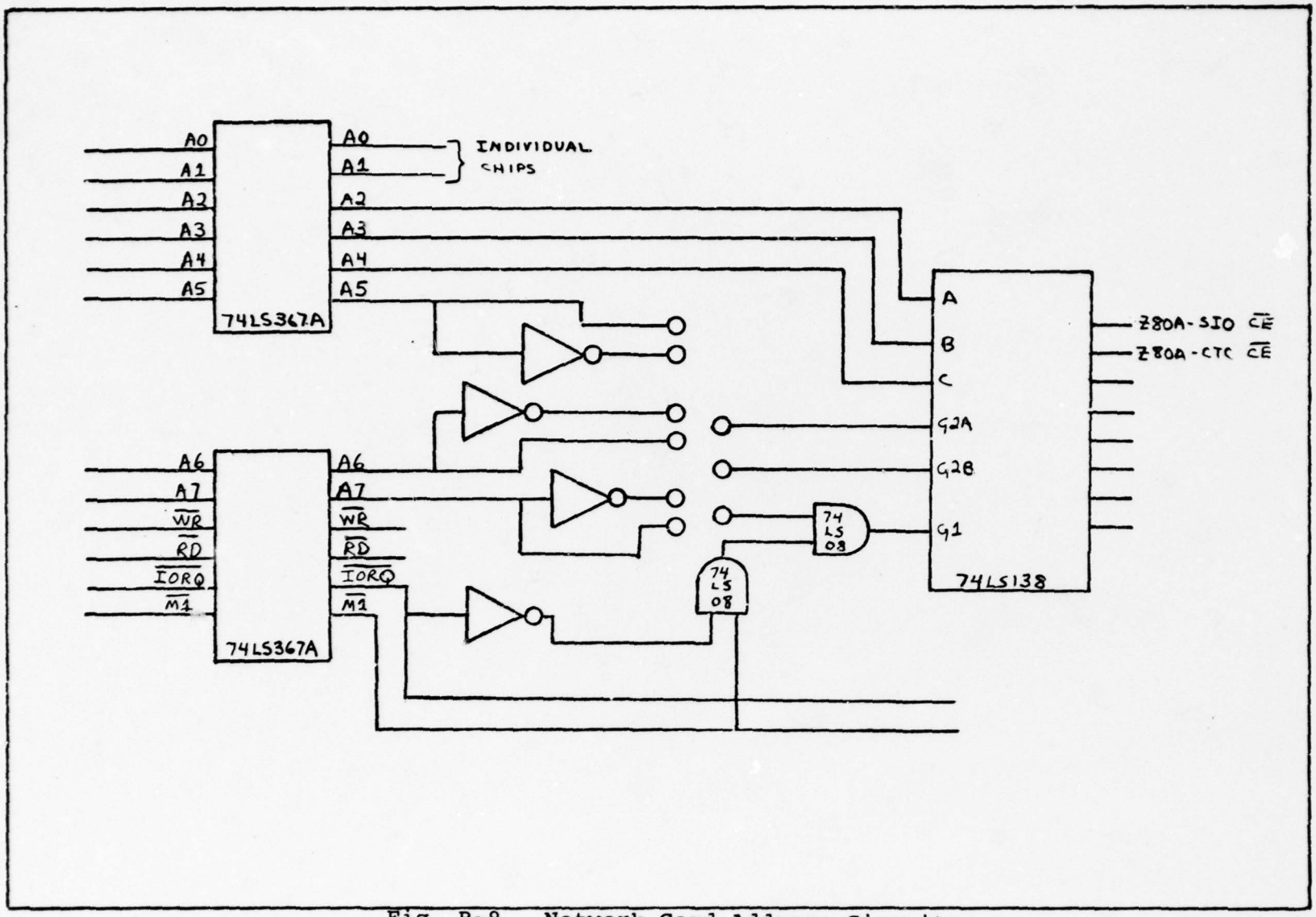


Fig. B-8. Network Card Address Circuitry

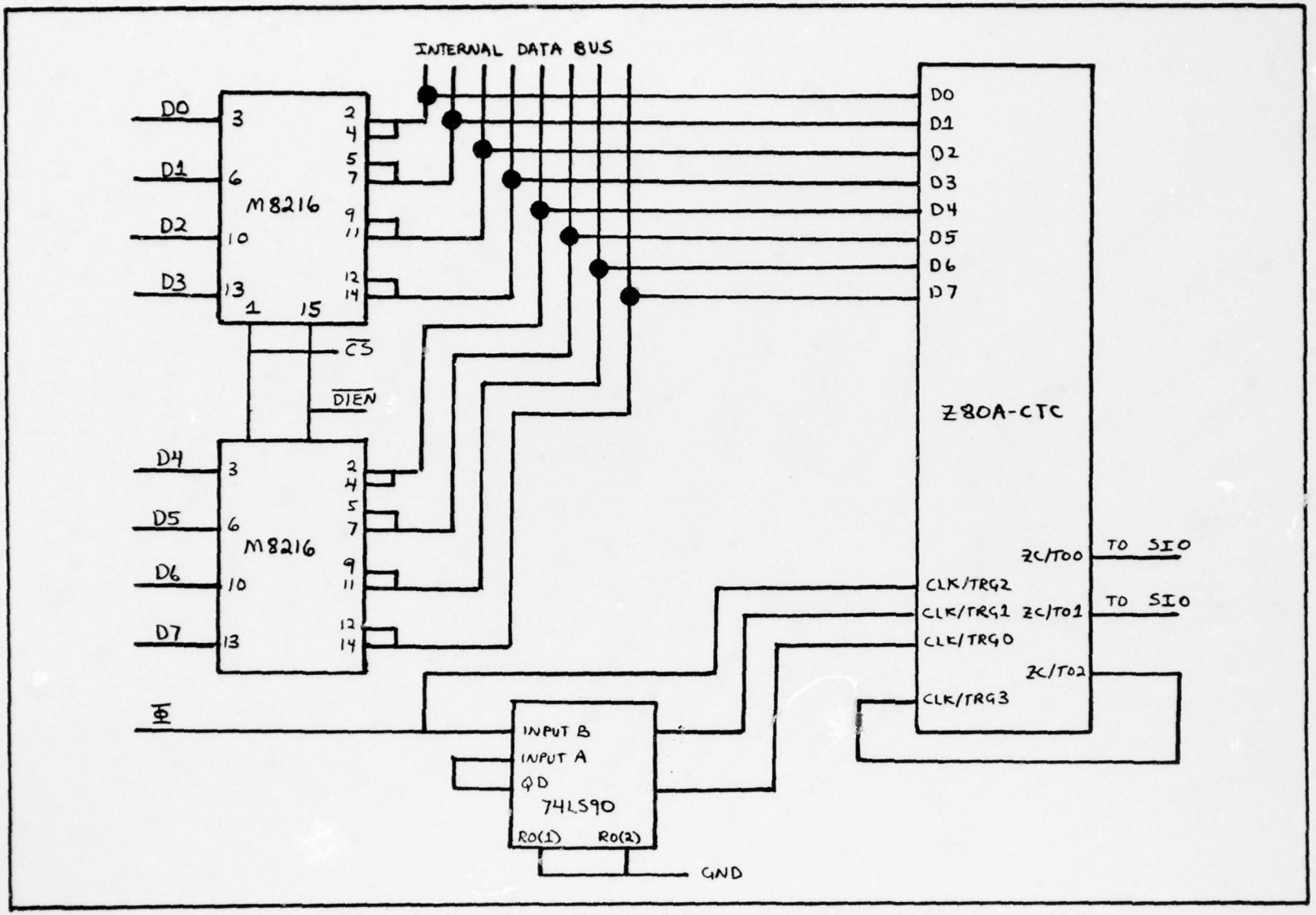


Fig. B-9. Network Card Data Bus Circuitry

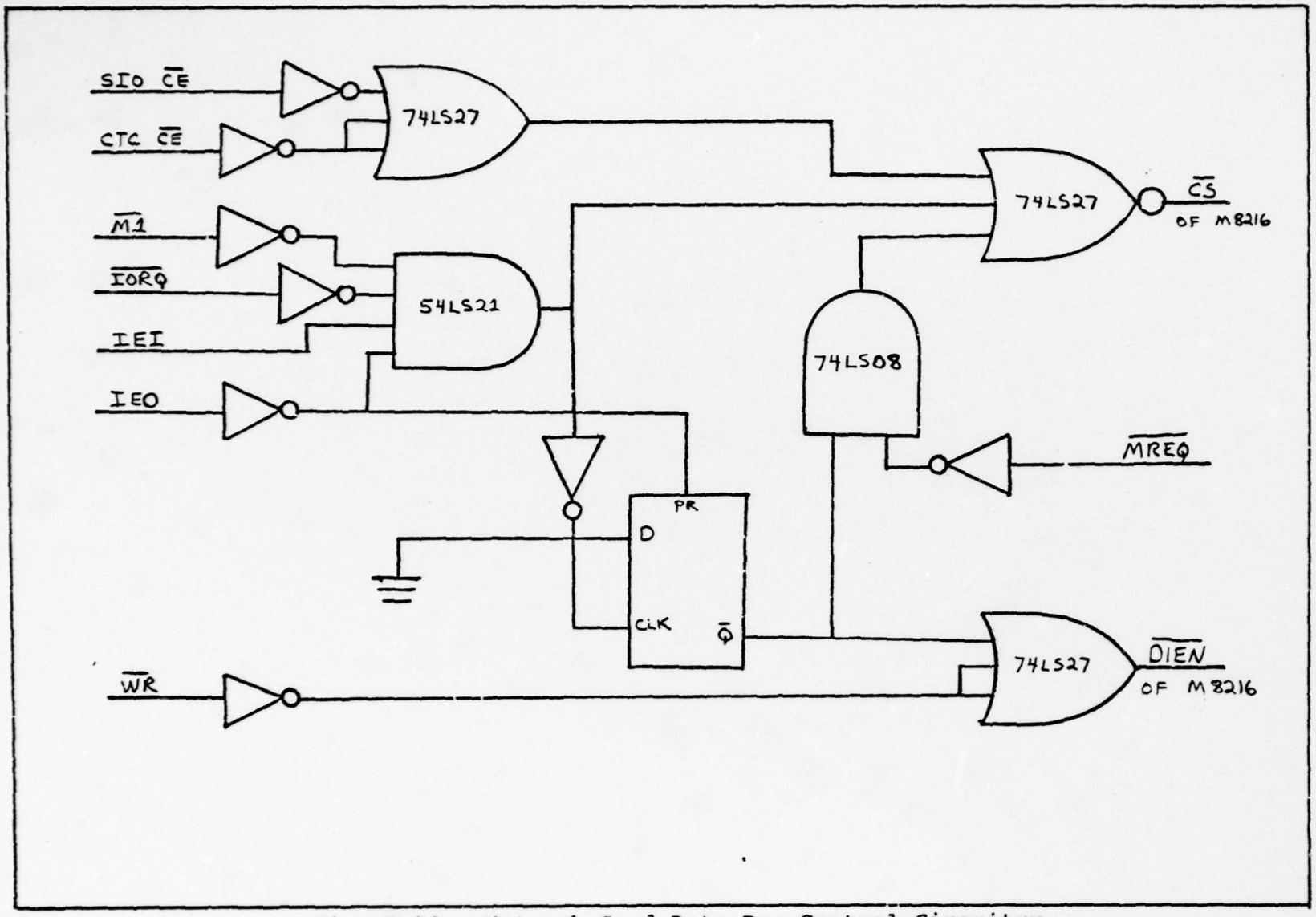


Fig. B-10. Network Card Data Bus Control Circuitry

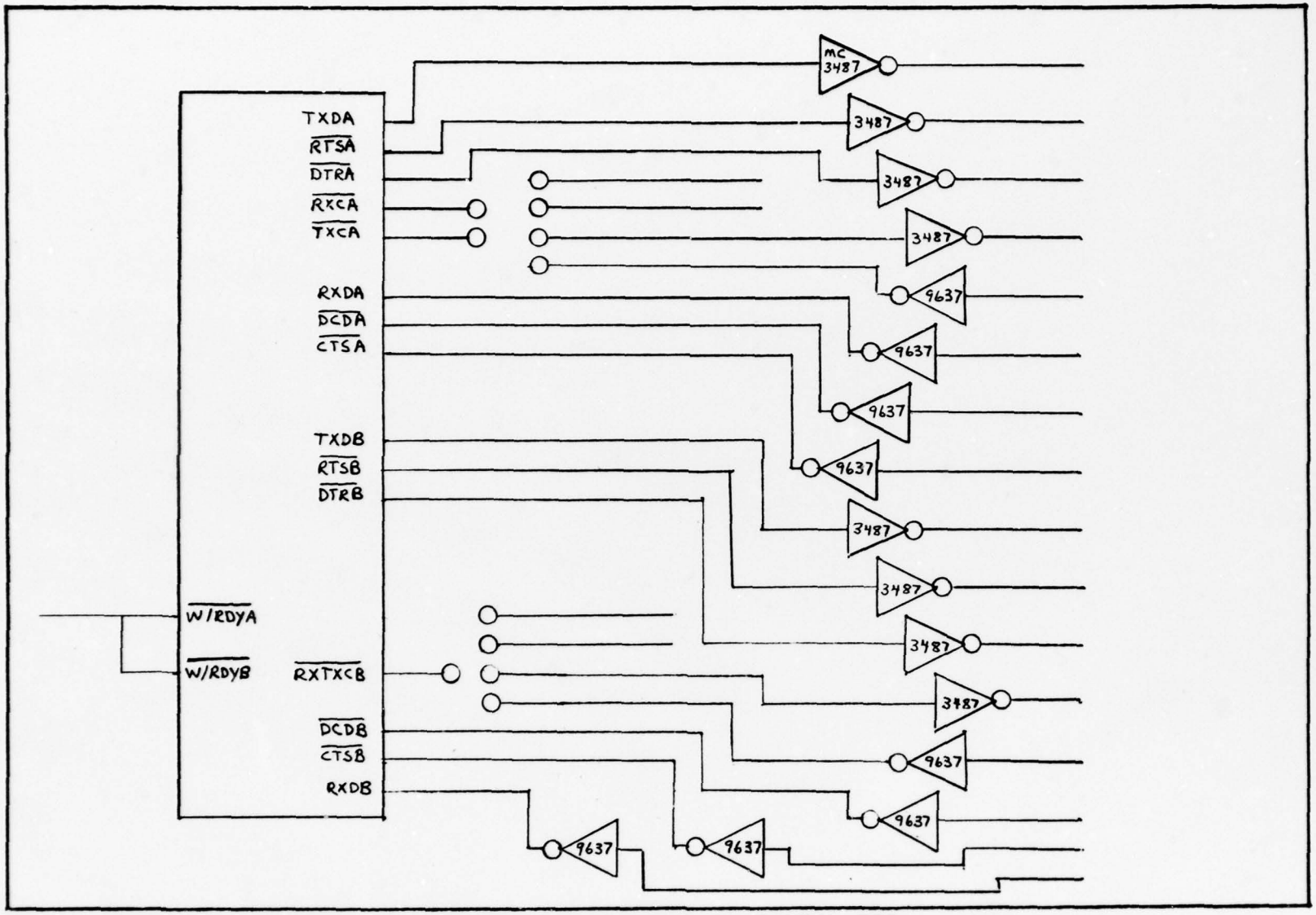


Fig. B-11. Network Card Z80A-SIO

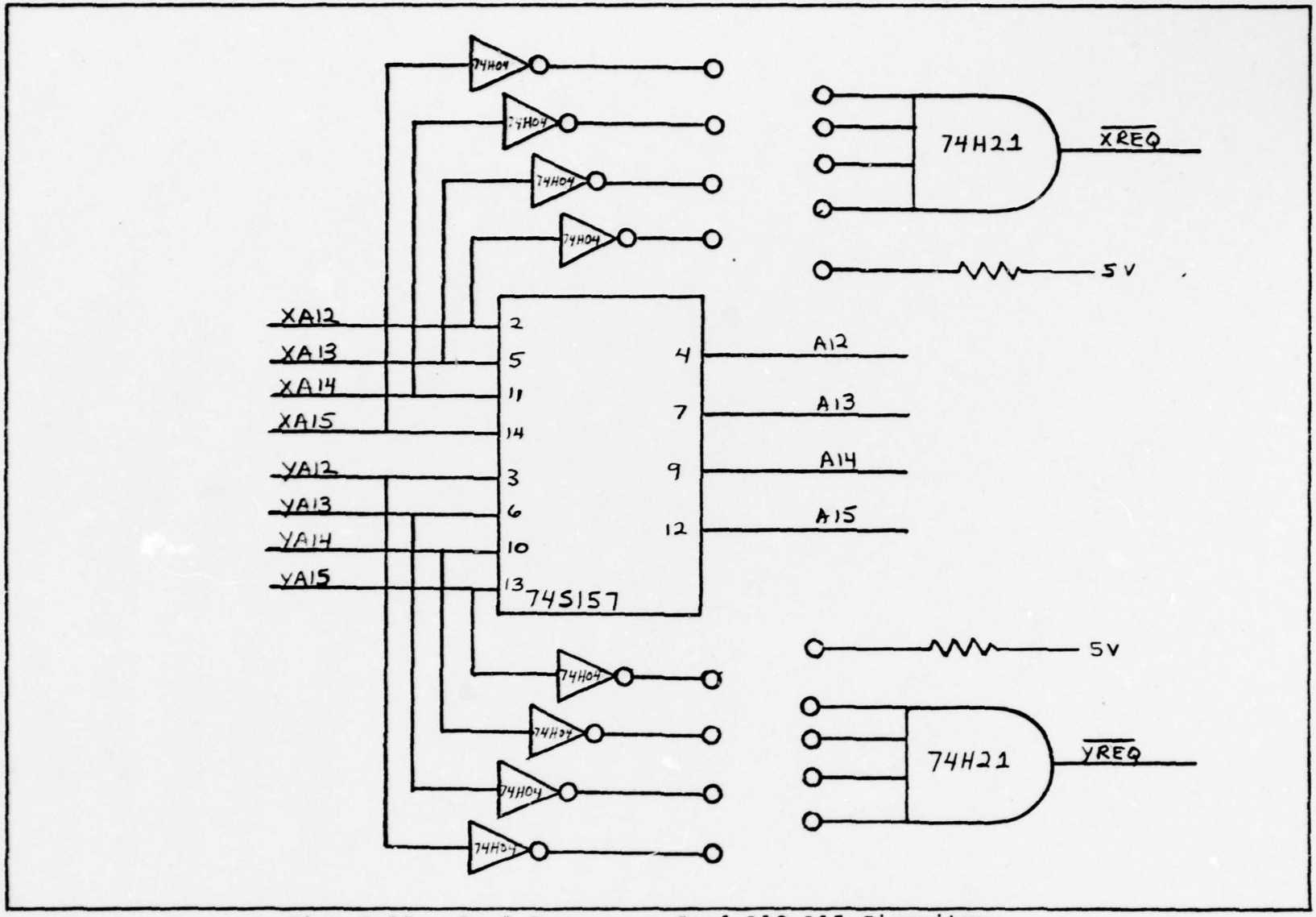


Fig. B-12. Dual Processor Card A12-A15 Circuitry

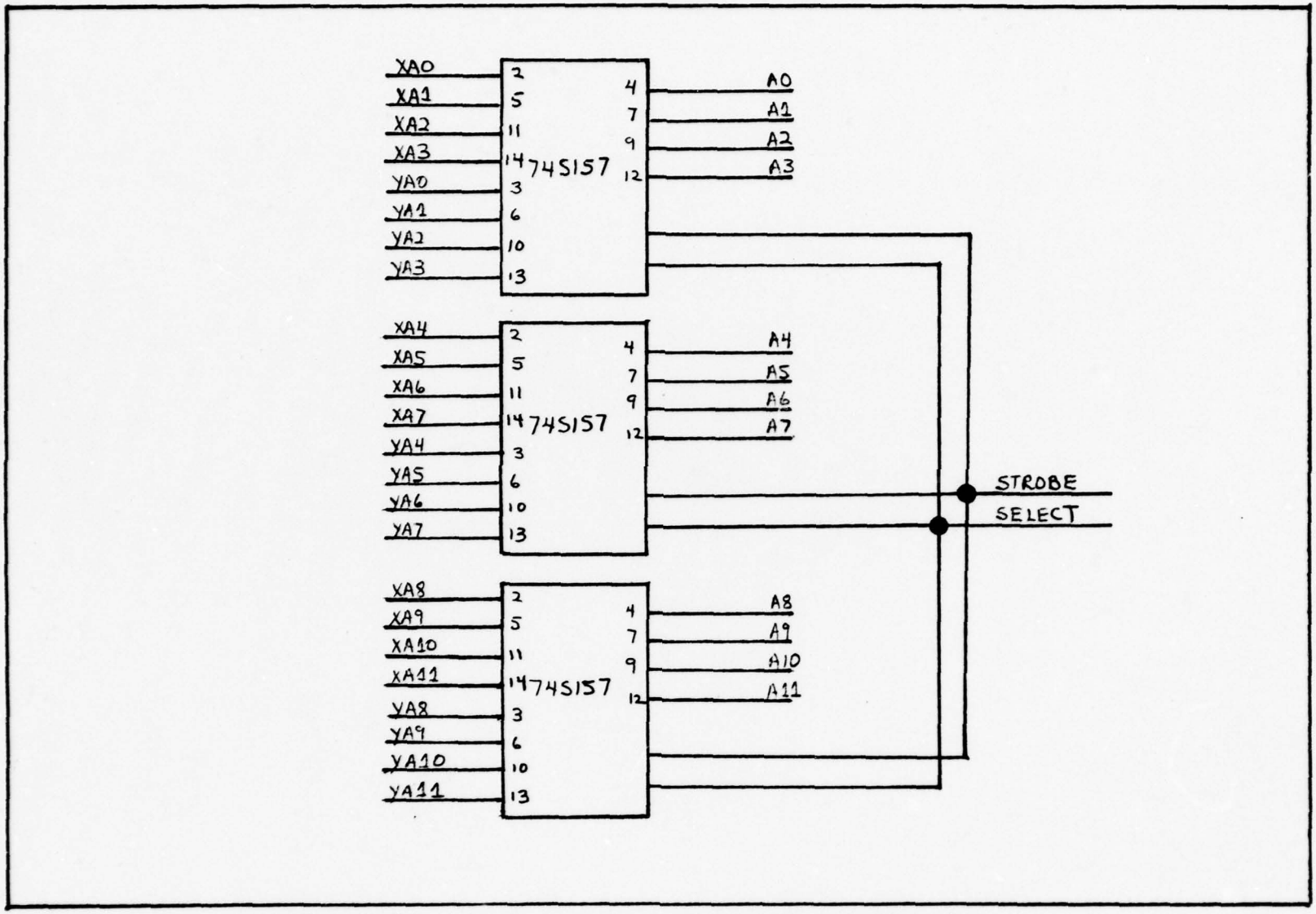


Fig. B-13. Dual Processor Card Address Circuitry

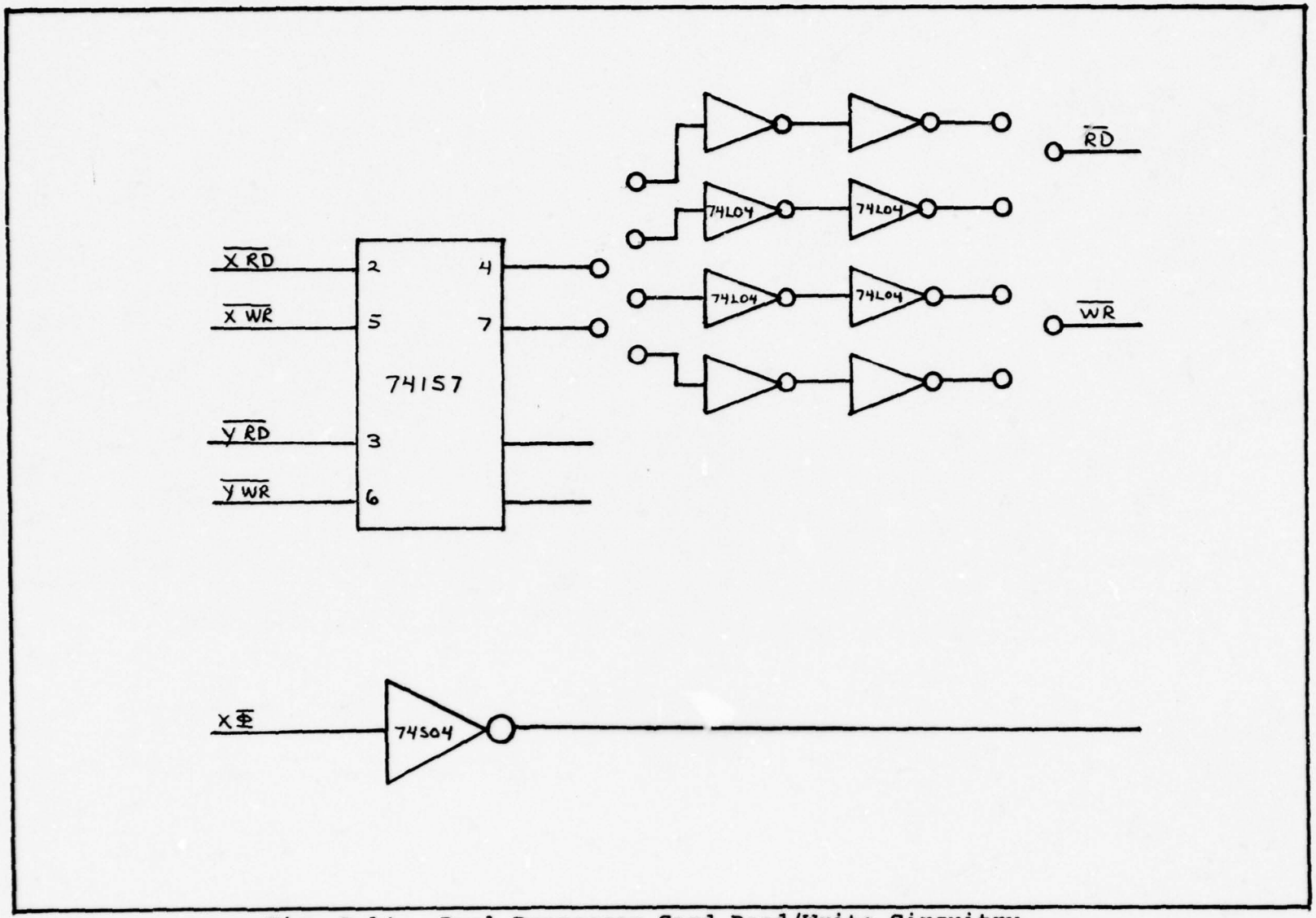


Fig. B-14. Dual Processor Card Read/Write Circuitry

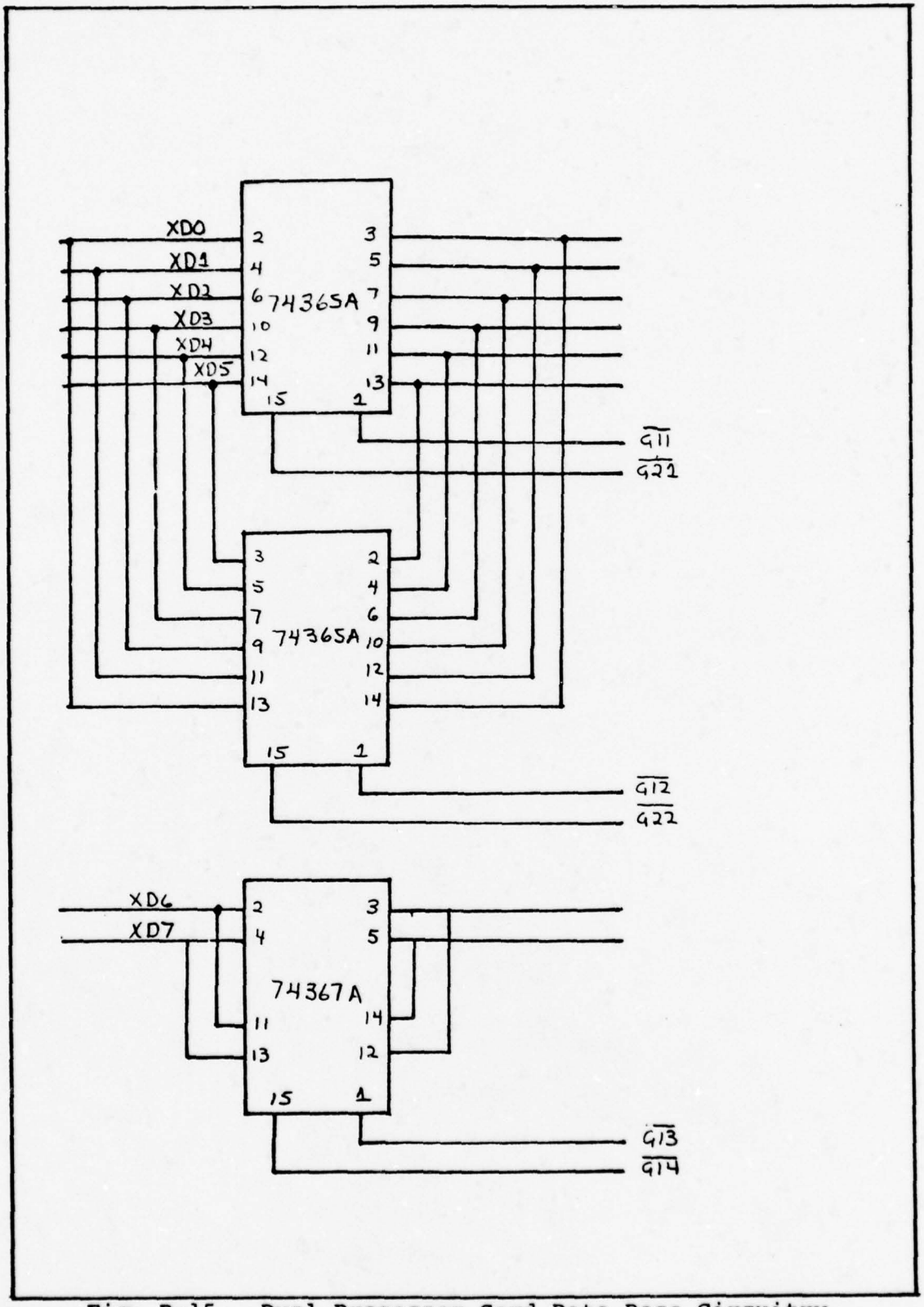


Fig. B-15. Dual Processor Card Data Base Circuitry

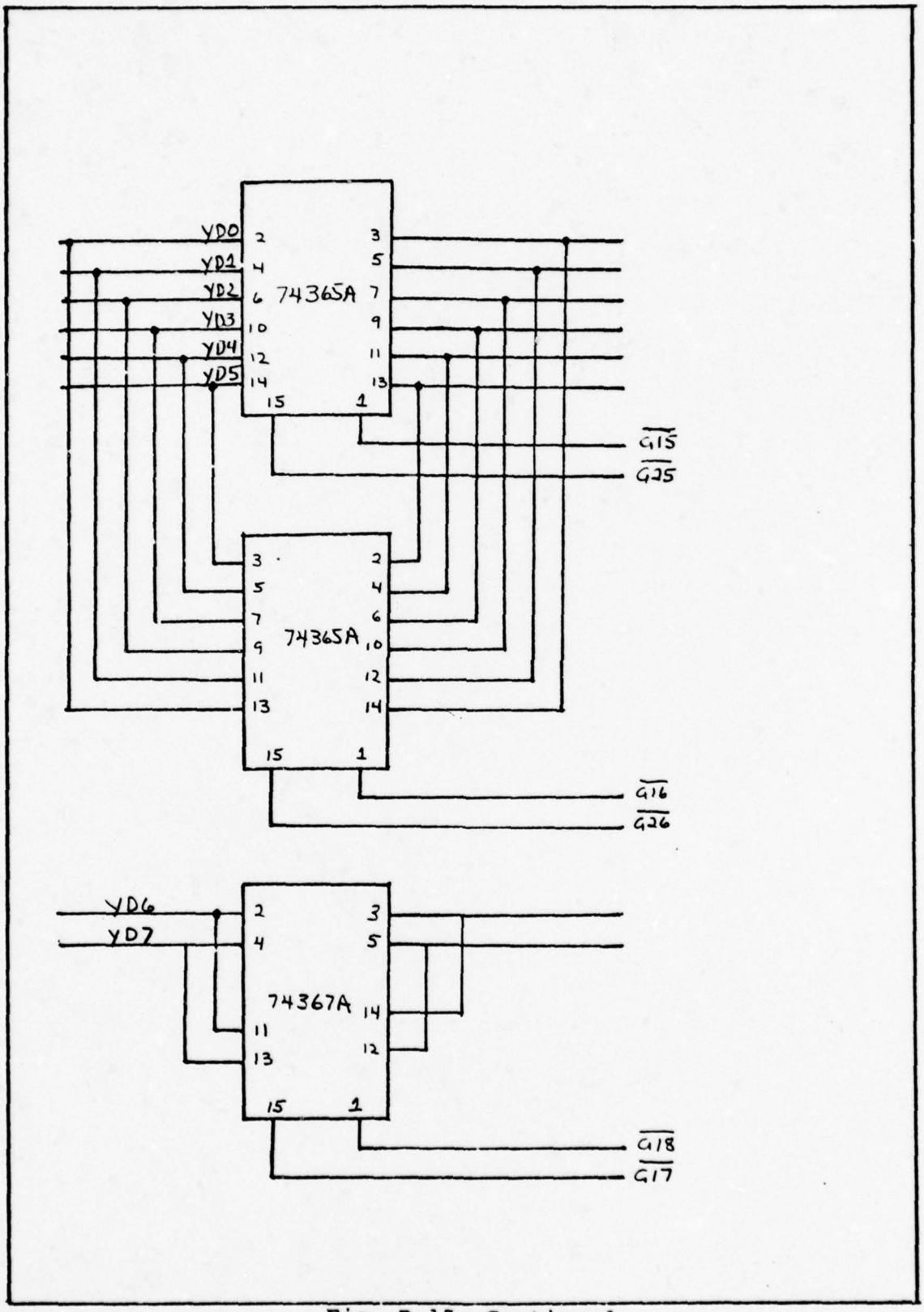


Fig. B-15--Continued

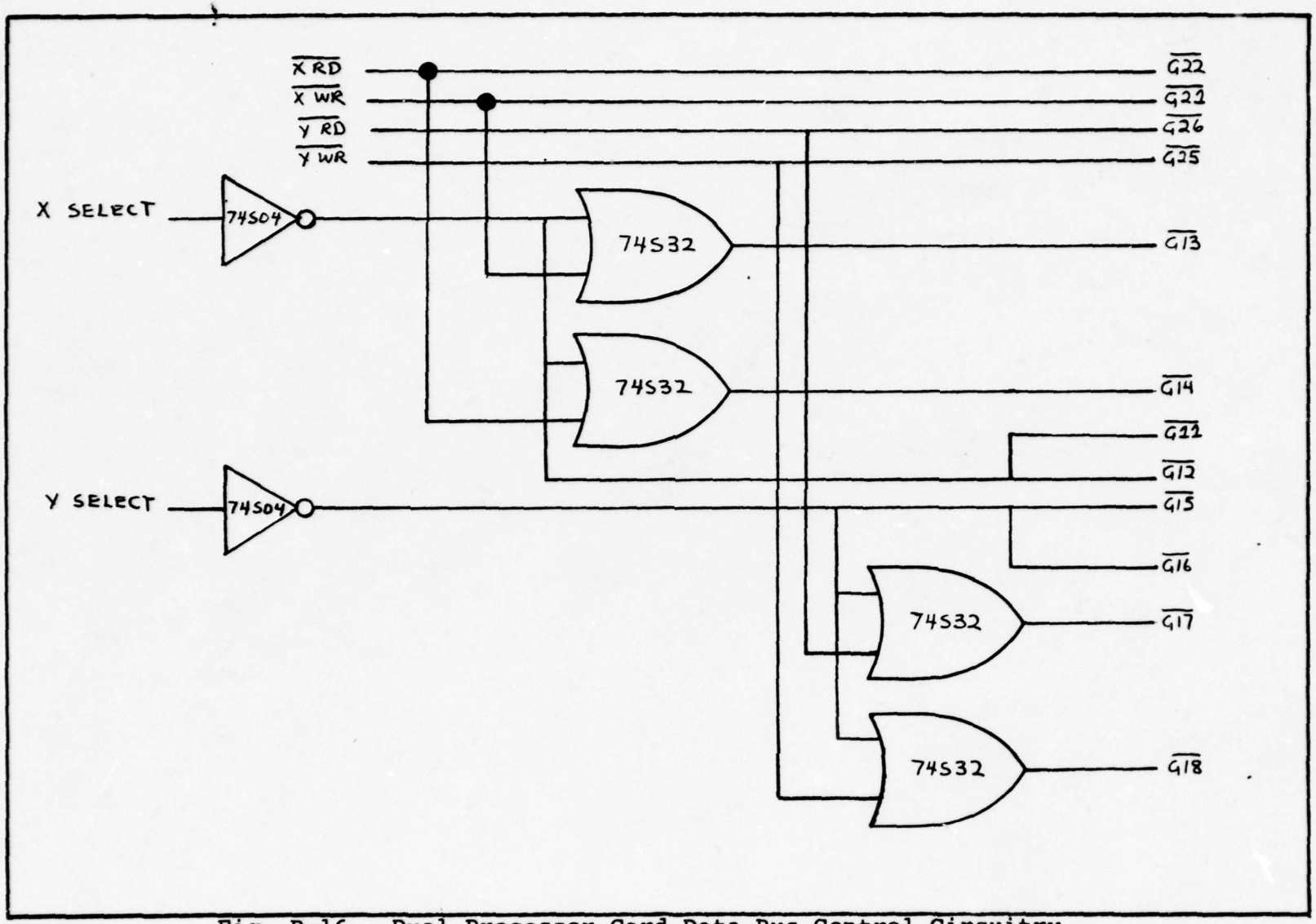


Fig. B-16. Dual Processor Card Data Bus Control Circuitry

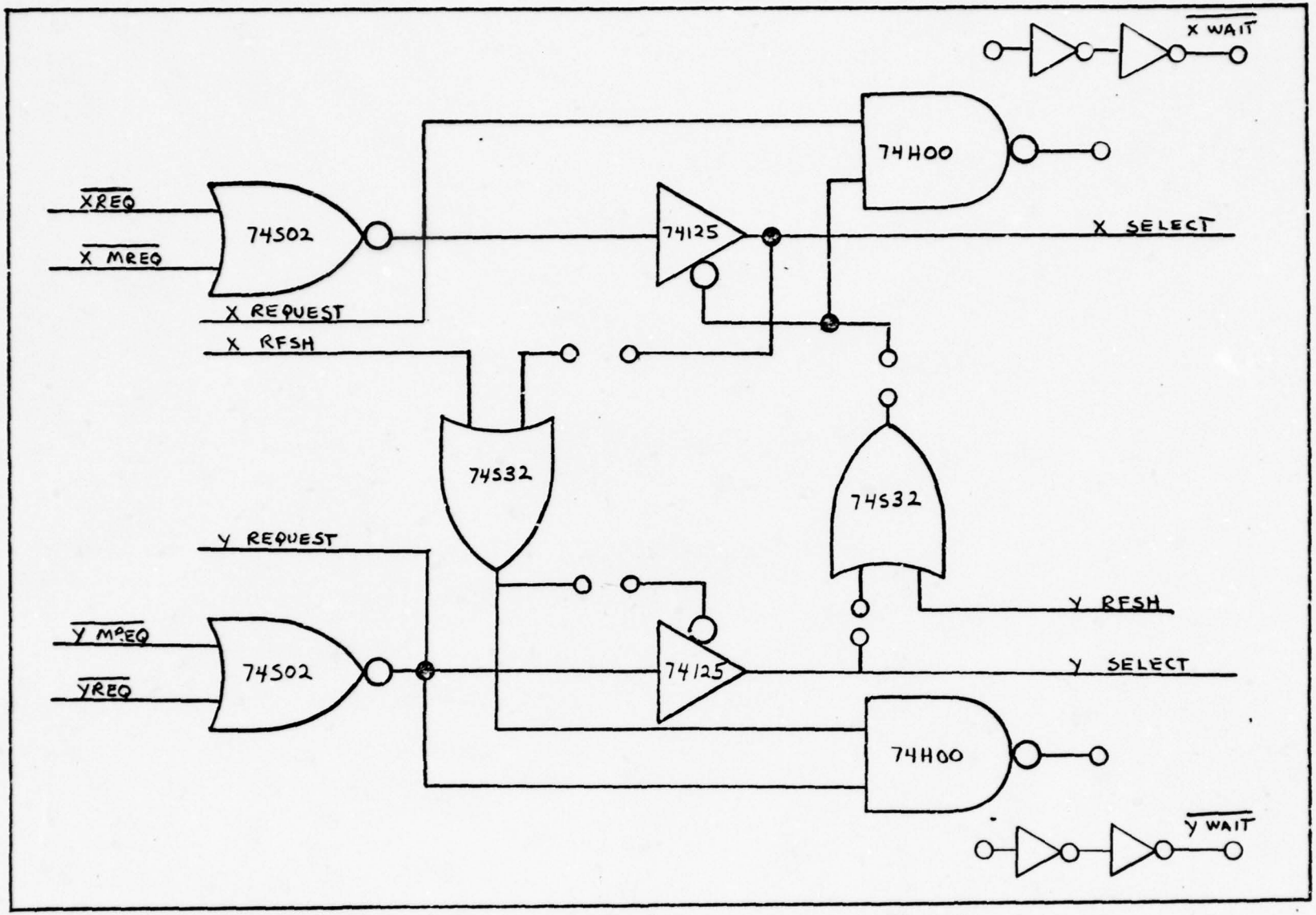


Fig. B-17. Dual Processor Card Memory Request Circuitry

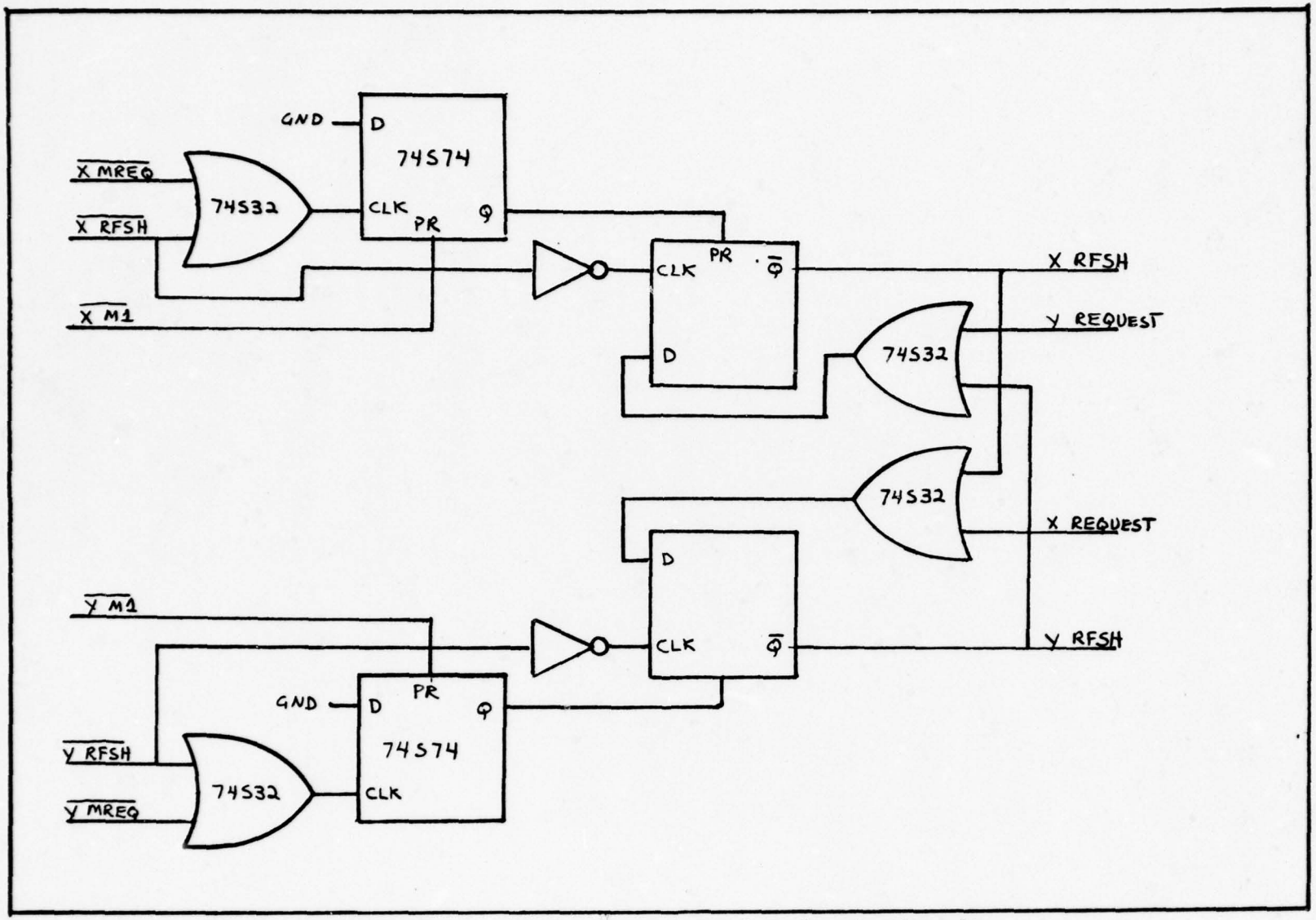


Fig. B-18. Dual Processor Card Refresh Circuitry

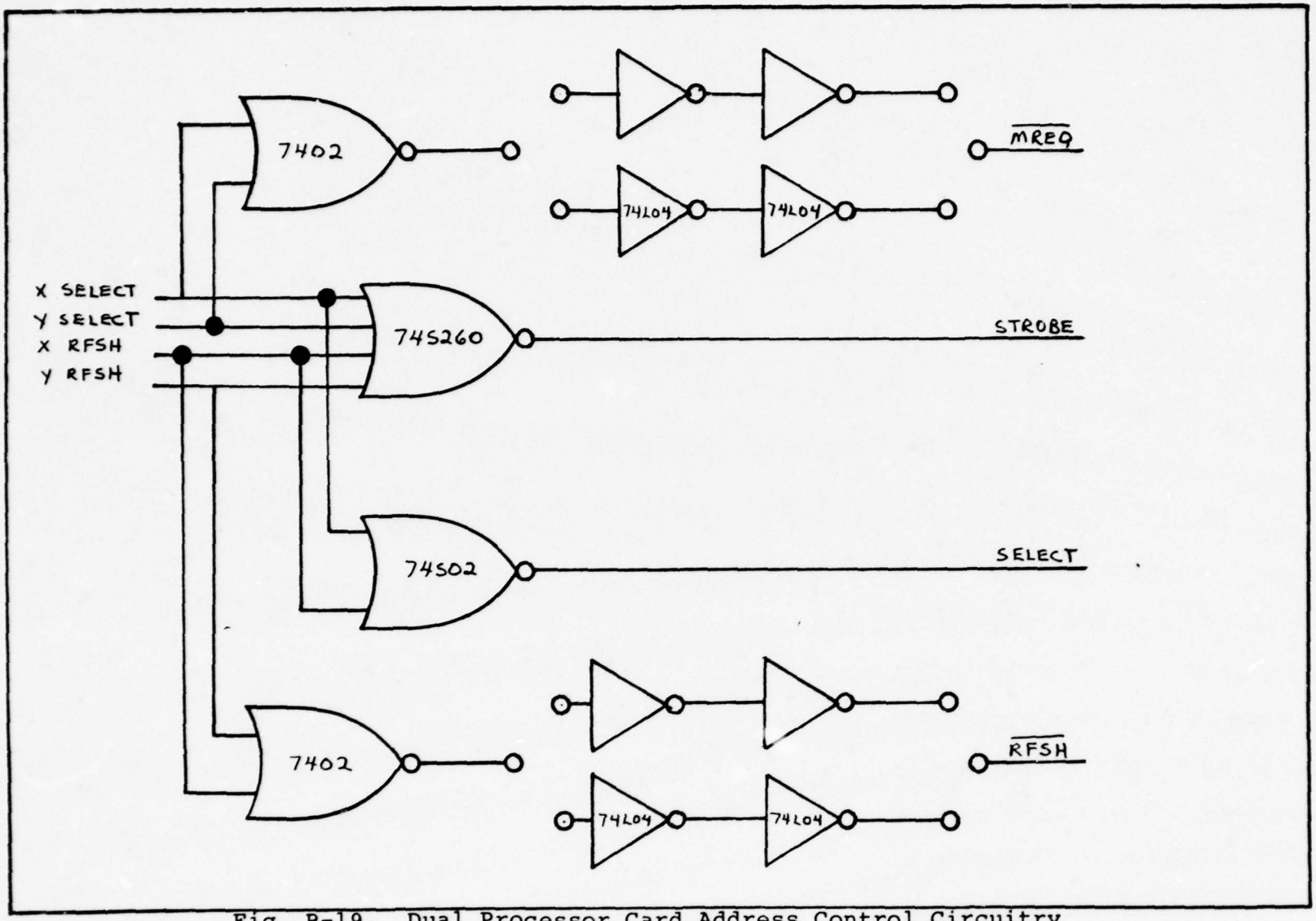


Fig. B-19. Dual Processor Card Address Control Circuitry

Appendix C

Assembled Software

This appendix provides a copy of the assembled versions of the two operating systems developed as a part of this investigation. A Mostek Z80 cross-assemblier was used to generate the object code. This assemblier was modified to allow it to operate on the CDC 6600.

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND      COMMENT

                  1  ;////////////////////////////////////////////////////////////
                  2  ;///////////////        OPERATING SYSTEM #1        ///////////////
                  3  ;////////////////////////////////////////////////////////////
                  4  ;THIS ROUTINE ESTABLISHES THE OPERATING SYSTEM FOR PROCESS-
                  5  ;OR #1 OF THE UNIVERSAL NETWORK INTERFACE. THIS PROCESSOR
                  6  ;IS ASSIGNED THE FUNCTIONS ASSOCIATED WITH INTERFACING THE
                  7  ;PERIPHERALS TO THE NETWORK INTERFACE. THE OPERATING SYSTEM
                  8  ;IS COMPOSED OF TWO MAJOR AREAS. THE FIRST AREA CONSISTS OF
                  9  ;A NUMBER OF SECTIONS WHICH INITIALIZE THE DIFFERENT CHIPS
                 10  ;ASSOCIATED WITH PROCESSOR BOARD #1 AND THE INPUT CARD/
                 11  ;CARDS. THE SECOND SECTION ACTUALLY ACCOMPLISHES THE INTER-
                 12  ;FACING FUNCTION. THE DIFFERENT AREAS AND SECTIONS ARE
                 13  ;EXPLAINED AS THEY ARE ENCOUNTERED WITHIN THE OPERATING
                 14  ;SYSTEM
                 15  ;
                 16  ;THIS SECTION INITIALIZES THE PROCESSOR BOARD CENTRAL  ******
                 17  ;PROCESSING UNIT. IT ESTABLISHES THE HIGH ORDER BYTE OF
                 18  ;THE VECTOR INTERRUPT TABLE (ITVTAB), THE LOCATION IN
                 19  ;MEMORY OF THE STACK POINTER (SPLOC) THE INTERRUPT
                 20  ;MODE FOR THE PROCESSOR AND THE VALUE OF THE REFRESH REG
0000 3E00        21          LD   A,00          ;A=ZERO
0002 ED4F        22          LD   R,A           ;SET REFRESH REG TO ZERO
0004 C34100      23          JP   ISTART        ;JUMP OVER RESTART AREA
                 24          ORG  65D
0041 21C00F      25  ISTART  LD   HL,INVTAB     ;HL=ADDRESS OF VECTOR ADDRESS TABLE
0044 7C          26          LD   A,H           ;A=HIGH BYTE VECTOR ADDRESS TABLE
0045 ED47        27          LD   I,A           ;I=HIGH BYTE VECTOR ADDRESS TABLE
0047 DD2AC010    28          LD   IX,(SPLOC)    ;IX=MEMORY ADDRESS OF STACK POINTER
0048 DDF9        29          LD   SP,IX         ;LOAD THE STACK POINTER WITH IX
004D ED56        30          IM   1             ;SET INTERRUPT MODE TO VECTOR ADD MODE
                 31  ;**** THIS SECTION INITIALIZES THE NON SYNCHRONOUS USARTS  *****
                 32  ;USING THE USART CHARACTERISTICS LINKED LIST. ALL NON
                 33  ;SYNCHRONOUS USART MUST BE INCLUDED WITHIN THE LINKED LIST
                 34  ;TO INSURE PROPER INITIALIZATION. THE FIRST ENTRY IN THE
                 35  ;LIST MUST BE FOR THE PROCESSOR BOARD USART. FOR EACH
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

                 36  ;USART THE FOLLOWING MUST BE PROVIDED IN ORDER INDICATED:
                 37  ;UARTXX   -I/O ADDRESS OF USART HOLDING REGISTER
                 38  ;       +1 -ADDRESS OF WORD USED TO HOLD MEMORY
                 39  ;       +2 -BLOCK ADDRESS FOR LOCAL MESSAGES
                 40  ;       +3 -COMMAND REGISTER WORD
                 41  ;       +4 -MODE REGISTER WORD
                 42  ;       +5 -MODE REGISTER WORD
                 43  ;       +6 -Z80A-CTC I/O CHANNEL ADDRESS FOR CHANNEL SUPPLY-
                 44  ;           ING THE FREQUENCY FOR THE USART
                 45  ;       +7 -Z80A-CTC CHANNEL MODE WORD
                 46  ;       +8 -Z80A-CTC PRESCALER VALUE
                 47  ;       +9 -ADDRESS OF PARAMETER LIST FOR NEXT NON
                 48  ;       +10-SYNCHRONOUS USART
                 49  ;WHERE UARTXX IS A LABEL FOR THE PARAMETER LIST . EACH LIST
                 50  ;MUST CONTAIN AN ENTRY FOR EACH CHARACTERISTIC. THE LAST ENTRY
                 51  ;IN THE LIST MUST HAVE ZERO IN LOCATION UARTXX+9 AND UARTXX+10
                 52  ;THE PROCESSOR BOARD USART PARAMETER LIST MUST BE LABELED
                 53  ;WITH UART00.
004F 21CC10      54          LD   HL,UART00     ;START LOCATION NON-SYN LINK LIST
0052 CD1603      55          CALL ITUART        ;CALL USART INITIALIZATION SUBROUTINE
0055 3ABF10      56          LD   A,(INVTAB+255D) ;LOW BYTE PROCESSOR BOARD VECTOR ADD
0058 D3DA        57          OUT  (218D),A      ;SET PIO PORT A INTERRUPT VECTOR TO A
005A 3ECF        58          LD   A,11001111B   ;SELECT BIT CHANGE INTIATATED INTERRUPT
005C D3DA        59          OUT  (218D),A      ;(MODE 3) FOR PORT A OF PROCESSOR BOARD
005E 3E80        60          LD   A,10000000B   ;SET BIT 7 OF PORT A PROCESSOR BOARD
0060 D3DA        61          OUT  (218D),A      ;PIO AS AN INPUT
0062 3E87        62          LD   A,10000111B   ;SET BIT 7 TO MONITOR
0064 D3DA        63          OUT  (218D),A      ;FOR A LOW LEVEL
0066 AF          64  NONSYN  XOR  A             ;SET A TO ZERO
0067 81          65          ADD  A,C           ;A=ZERO+LOW BYTE ADDRESS OF NEXT ENTRY
0068 80          66          ADD  A,B           ;A=A + HIGH BYTE OF NEXT ENTRY IN LIST
0069 CA7400      67          JP   Z,SYNC        ;JUMP TO SYNCHRONOUS USART ROUTINE
006C 69          68  NONSY1  LD   L,C           ;HL=ADDRESS OF NEXT NON-SYNCHRONOUS
006D 60          69          LD   H,B           ;USART PARAMETER TABLE
006E CD1603      70          CALL ITUART        ;CALL USART INITIALIZATION SUBROUTINE
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

0071 C36600     71           JP   NONSYN        ;REPEAT THE LOOP
                72  ;*** THIS SECTION INITIALIZES THE SYNCHRONOUS USARTS USING *****
                73  ;THE USART CHARACTERISTICS LINKED LIST. ALL SYNCHRONOUS
                74  ;USART MUST BE INCLUDED WITHIN THE LINKED LIST TO INSURE
                75  ;PROPER INITIALIZATION. THE FIRST SYNCHRONOUS USART IN THE
                76  ;TABLE CANNOT BE ASSIGNED I/O PORT ADDRESS ZERO SINCE THIS
                77  ;SIGNIFIES NO TABLE ENTRIES. EACH SYNCHRONOUS USART PARA-
                78  ;METER LIST MUST CONTAIN THE SAME ENTRIES AS FOR THE NON
                79  ;SYNCHRONOUS CASE PLUS THE FOLLOWING ADDITIONS:
                80  ;       +11-FIRST SYNCHRONIZATION CHARACTER
                81  ;       +12-SECOND SYNCHRONIZATION CHARACTER
                82  ;       +13-THE DELETE CHARACTER
                83  ;EACH LIST MUST BE LABELED AND CONTAIN AN ENTRY FOR
                84  ;EACH PARAMETER. THE FIRST ENTRY IN THE LIST MUST BE
                85  ;LABELED WITH SART00
0074 21F210     86  SYNC    LD   HL,SART00    ;HL=START LOCATION SYN USART LINK LIST
0077 AF         87  SYNC2   XOR  A            ;A=0
0078 86         88           ADD  A,(HL)       ;A=CONTENTS OF LOCATION SART00
0079 CA9A00     89           JP   Z,ENDSYN     ;IF ZERO NO ENTRIES IN LINK LIST
007C E5         90           PUSH HL           ;SAVE START OF PARAMETER LIST ADDRESS
007D CD1603     91           CALL ITUART       ;CALL USART INITIALIZATION ROUTINE
0080 E1         92           POP  HL           ;HL=START OF PARAMETER LIST ADDRESS
0081 C5         93           PUSH BC           ;SAVE NEXT PARAMETER LIST ADDRESS
0082 4E         94           LD   C,(HL)       ;C=I/O ADDRESS OF SYN USART HOLDING REG
0083 0C         95           INC  C            ;C=I/O ADDRESS FOR SYN CHARACTER REG
0084 110300     96           LD   DE,11D       ;DE=11
0087 19         97           ADD  HL,DE        ;HL=PARAMETER LIST ADDRESS OF SYN CHAR
0088 EDA3       98           OUTI              ;OUTPUT SYN CHARACTER #1
008A EDA3       99           OUTI              ;OUTPUT SYN CHARACTER #2
008C EDA3      100           OUTI              ;OUTPUT DLE CHARACTER
008E C1        101           POP  BC           ;BC=NEXT PARAMETER LIST ADDRESS
008F AF        102           XOR  A            ;A=0
0090 80        103           ADD  A,B          ;DETERMINE IF THERE IS
0091 81        104           ADD  A,C          ;ANOTHER ENTRY IN THE LINKED LIST
0092 CA9A00    105           JP   Z,ENDSYN     ;IF ZERO JUMP TO END OF SECTION
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND      COMMENT

0095 69         106  SYNC1   LD   L,C          ;HL=ADDRESS OF NEXT SYNCHRONOUS
0095 60         107          LD   H,B          ;USART PARAMETER TABLE
0097 C37700     108          JP   SYNC2        ;REPEAT THE LOOP
                109  ;****THIS SECTION INITIALIZES THE PRIORITY INTERRUPT      ******
                110  ;CONTROLLERS THROUGH THE USE OF THE PARAMETER LIST. THE
                111  ;PARAMETER LIST MUST CONTAIN THE FOLLOWING FOR EACH
                112  ;PRIORITY INTERRUPT CONTROLLER.
                113  ;PICXX    -I/O ADDRESS OF PIC
                114  ;       +1 -PIC MASK WORD
                115  ;       +2 -ADDRESS OF NEXT PRIORITY INTERRUPT
                116  ;       +3 -CONTROLLER PARAMETER LIST
009A 21C510     117  ENDSYN  LD   HL,PIC1     ;HL=START LOCATION PIC LINK LIST
009D 4E         118  PICIB   LD   C,(HL)      ;C=I/O ADDRESS PIC'S Z80A-CTC
009E 23         119          INC  HL           ;HL=ADDRESS OF MASK WORD
009F 46         120          LD   B,(HL)       ;B=MASK WORD
00A0 ED41       121          OUT  (C),B        ;OUTPUT MASK WORD
00A2 23         122          INC  HL           ;HL=ADD OF START OF PARAMETER LIST + 2
00A3 AF         123          XOR  A            ;A=ZERO
00A4 4E         124          LD   C,(HL)       ;C=LOW BYTE OF ADD NEXT PARAMETER LIST
00A5 81         125          ADD  A,C          ;A=LOW BYTE OF ADDR NEXT PAPAMETER LIST
00A6 C2AF00     126          JP   NZ,PICIA     ;IF NOT ZERO HAVE ANOTHER LIST SO JUMP
00A9 23         127          INC  HL           ;HL=ADD OF START OF PARAMETER LIST + 3
00AA 46         128          LD   B,(HL)       ;B=HIGH BYTE OF ADDRESS NEXT PARAMETER
00AB 80         129          ADD  A,B          ;A=B
00AC CA3500     130          JP   Z,ENDPIC     ;IF ZERO NO MORE LIST SO JUMP
00AF 69         131  PICIA   LD   L,C          ;HL=ADDRESS OF NEXT PRIORITY INTERRUPT
00B0 60         132          LD   H,B          ;CONTROLLER PARAMETER LIST
00B1 C39D00     133          JP   PICIB        ;REPEAT THE LOOP
                134  ;****THIS SECTION INITIALIZES PROCESSOR BOARD PARALLEL INPUT ****
                135  ;/OUTPUT CHIP (PIO). PORT A OF PIO IS USED FOR PROCESSOR
                136  ;BOARD USART INTERRUPT DETECTION AND HAS BEEN PREVIOUSLY
                137  ;INITIALIZED. THIS SECTION INITIALIZES PORT B OF THE PIO.
00B4 E1         138          POP  HL           ;HL=ADD OF CURRENT TAIL OF QUEUE
                139  ;THIS PORT CANNOT BE CONFIGURED FOR MODE TWO (BI-DIRECTIONAL
                140  ;I/O PORT). IN ADDITION AN ACTIVE PORT B SHOULD NOT BE AS-
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND      COMMENT

               141   :SIGNED I/O PORT ADDRESS ZERO SINCE THIS ADDRESS IS USED TO
               142   :SIGNIFY AN INACTIVE PORT B. THE PARAMETER LIST FOR PORT B
               143   :HAS BEEN ASSIGNED ADDRESS PIO1 AND CONSISTS OF THE FOLLOWING:
               144   :PIO1      -I/O ADDRESS OF PORT B PIO
               145   :       +1 -PORT B MODE WORD
               146   :       +2 -PORT B LOW BYTE VECTOR ADDRESS
               147   :IF PORT B IS NOT BEING USED THEN IT SHOULD BE ASSIGNED
               148   :I/O ADDRESS ZERO
0085 21EF10    149   ENDPIC  LD   HL,PIO1      :HL=START OF PORT B PARAMETER LIST
0088 AF        150           XOR  A             :A=ZERO
0089 BE        151           CP   (HL)          :COMPARE I/O ADDRESS TO ZERO
008A CAC300    152           JP   Z,ENDPIO      :IF ZERO PORT B NOT USED SO JUMP
008D 4E        153           LD   C,(HL)        :C=PORT B I/O ADDRESS
008E 23        154           INC  HL            :HL=START OF PARAMETER LIST + 1
008F EDA3      155           OUTI               :OUTPUT MODE WORD
0091 EDA3      156           OUTI               :OUTPUT LOW BYTE VECTOR ADDRESS
               157   :****THIS SECTION SET UP THE MEMORY TABLE USED FOR ALLOCATION****
               158   :DEALLOCATION OF MEMORY BLOCKS. THESE MEMORY BLOCKS ARE
               159   :USED FOR MESSAGE STORAGE AND ARE FIXED LENGTH BLOCKS.
               160   :THE LENGTH,NUMBER,AND LOCATION OF THE BLOCKS ARE A
               161   :USER INPUTTED ITEM ALONG WITH THE LOCATION OF THE MEMORY
               162   :TABLE.
               163   :THE FOLLOWING VARIABLES ARE USED BY THIS SECTION AND THE
               164   :VALUE OF THESE VARIABLES MUST BE SPECIFIED BY THE USER
               165   :THE LIMITATION PLACED ON THESE ARE THE BLOCK SIZE CANNOT
               166   :EXCEED 256D AND THE MEMORY BLOCKS MUST BE IN CONTIGUOUS
               167   :MEMORY. THE OTHER VARIABLE ASSOCIATED WITH THIS SECTION
               168   :IS MNTBPT WHICH IS THE ADDRESS OF THE HEAD OF THE MEMORY
               169   :TABLE. THE MEMORY TABLE CONSISTS OF THE ADDRESS OF ALL
               170   :UNALLOCATED MEMORY BLOCKS.
               171   :          BLKNUM - NUMBER OF MEMORY BLOCKS
               172   :          BLKSIZ - SIZE OF EACH MEMORY BLOCK
               173   :          LOMNTB - LOCATION OF MEMORY TABLE
               174   :          MNTBED-END OF MEMORY TABLE ADDRESS
               175   :          MEMST  - START LOCATION FOR THE MEMORY BLOCKS
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

00C3 ED4B0111   176  ENDPIO   LD   BC,(BLKNUM)   ;BC=NUMBER OF MEMORY BLOCKS
00C7 ED590011   177           LD   DE,(BLKSIZ)   ;DE=MEMORY BLOCK SIZE
00CB DD21431F   178           LD   IX,LOMNTB     ;IX=LOCATION FOR MEMORY TABLE
00CF DD22411F   179           LD   (MNTBPT),IX   ;MEMORY POINTER TO START OF TABLE ADD
00D3 2A0211     180           LD   HL,(MENST)    ;HL=START ADDRESS FOR THE MEMORY BLOCKS
00D6 DD7500     181  ITMEN    LD   (IX+0),L      ;MEMORY TABLE=LOW BYTE MEMORY BLOCK ADD
00D9 DD7401     182           LD   (IX+1),H      ;MEMORY TABLE=HGH BYTE MEMORY BLOCK ADD
00DC DD23       183           INC  IX            ;INCREMENT IX TO NEXT LOCATION IN
00DE DD23       184           INC  IX            ;THE MEMORY TABLE
00E0 19         185           ADD  HL,DE         ;HL=BLOCK ADDRESS + BLOCK SIZE
00E1 0B         186           DEC  BC            ;BC=NUMBER OF BLOCKS - 1
00E2 C2D600     187           JP   NZ,ITMEN      ;REPEAT IF NUMBER OF BLOCKS NOT=TO ZERO
                188  ;****THIS SECTION INITIALIZES THE DIFFERENT QUEUES USED BY ******
                189  ;THE INPUT BOARD PROCESSOR. THESE QUEUES ARE AS FOLLOWS
                190  ;          LOTXSQ  -LOCAL TRANSMIT QUEUE START ADDRESS
                191  ;          LOTXEQ  -LOCAL TRANSMIT QUEUE END ADDRESS
                192  ;          LBTXSQ  -LOCAL TRANSMIT BUSY QUEUE START ADDRESS
                193  ;          LBTXEQ  -LOCAL TRANSMIT BUSY QUEUE END ADDRESS
                194  ;OTHER LABELS ASSOCIATED WITH THE QUEUES ARE
                195  ;          LOTXHQ  -ADDRESS OF HEAD OF LOCAL TRANSMIT QUEUE
                196  ;          LOTXTQ  -ADDRESS OF TAIL OF LOCAL TRANSMIT QUEUE
                197  ;          LBTXHQ  -ADDRESS OF HEAD OF LOCAL TRANSMIT BUSY QUEUE
                198  ;          LBTXTQ  -ADDRESS OF TAIL OF LOCAL TRANSMIT BUSY QUEUE
                199  ;EACH QUEUE IS CIRCULAR IN NATURE. THE OTHER
                200  ;QUEUE USED BY THE INPUT PROCESSOR IS THE NETWORK TRANSMIT
                201  ;QUEUE. THIS QUEUE IS INITIALIZED BY THE NETWORK PROCESSOR.
                202  ;IT'S LABELS ARE
                203  ;          NWTXSQ  -NETWORK TRANSMIT QUEUE START ADDRESS
                204  ;          NWTXEQ  -NETWORK TRANSMIT QUEUE END ADDRESS
                205  ;          NWTXHQ  -NETWORK TRANSMIT QUEUE HEAD ADDRESS
                206  ;          NWTXTQ  -NETWORK TRANSMIT QUEUE TAIL ADDRESS
                207  ;THE INITIALIZATION ROUTINE SETS THE HEAD AND TAIL OF THE
                208  ;QUEUE TO THE START OF QUEUE ADDRESS
00E5 214F20     209           LD   HL,LOTXSQ     ;HL=ADDRESS OF START OF LOCAL TX QUEUE
00E8 224820     210           LD   (LOTXHQ),HL   ;SET HEAD OF LOCAL TX QUEUE TO START
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

00EB 224D20     211          LD   (LOTXTQ),HL   ;SET TAIL OF LOCAL TX QUEUE TO START
00EE 214938     212          LD   HL,LBTXSQ     ;HL=ADD OF START OF LOCAL BUSY TX QUEUE
00F1 224538     213          LD   (LBTXHQ),HL   ;SET HEAD OF LOCAL TX BUSY TO START
00F4 224738     214          LD   (LBTXTQ),HL   ;SET TAIL OF LOCAL TX BUSY TO START
                215  ;
                216  ;*************************************************************
                217  ;*************              MACRO DEFINITION          **********
                218  ;*************************************************************
                219  ;THIS SECTION DEFINES THE MACROS ADDTQ AND SUBHQ.
                220  ;THESE MACROS ARE USED TO ADD A MEMORY BLOCK ADDRESS TO
                221  ;THE TAIL OF A QUEUE(ADDTQ)AND TO REMOVE A MEMORY BLOCK
                222  ;ADDRESS FROM THE HEAD OF A QUEUE(SUBHQ). THE VARIABLES
                223  ;ASSOCIATED WITH THE MACROS ARE
                224  ;          #START-ADDRESS OF THE START OF THE QUEUE
                225  ;          #END  -ADDRESS OF THE END OF THE QUEUE
                226  ;          #HEAD -POINTER TO THE HEAD OF THE QUEUE
                227  ;          #TAIL -POINTER TO THE TAIL OF THE QUEUE
                228          LIST M
                229  ADDTQ   MACR #START,#END,#TAIL
                230          LD   HL,(#TAIL)    ;HL=ADDRESS OF TAIL OF QUEUE
                231          LD   (HL),C        ;PUT LOW ORDER BYTE OF THE MESSAGE
                232  ;                           MEMORY BLOCK AT TAIL OF QUEUE
                233          INC  HL            ;PUT HIGH ORDER BYTE OF THE MESSAGE
                234          LD   (HL),B        ;MEMORY BLOCK AT TAIL OF QUEUE
                235          INC  HL            ;HL=NEW TAIL OF QUEUE
                236          PUSH HL            ;SAVE NEW TAIL OF QUEUE ADDRESS
                237          LD   DE,#END       ;DE=ADDRESS OF END OF QUEUE
                238          SBC  HL,DE         ;HL=CURRENT LOCATION-END QUEUE-CARRY
                239          JP   M,A_#$YM      ;IF SUBTRACTION NEGATIVE JUMP
                240          LD   HL,#START     ;HL=ADDRESS OF START OF QUEUE
                241  A_#$YM  LD   (#TAIL),HL    ;STORE VALUE OF HL INTO TAIL ADDRESS
                242          ENDM
                243  SUBHQ   MACR #START,#END,#HEAD
                244          LD   HL,(#HEAD)    ;HL=ADDRESS OF HEAD OF QUEUE
                245          LD   C,(HL)        ;PUT LOW ORDER BYTE OF ADDRESS AT
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

                246  :                           HEAD OF QUEUE INTO REG C
                247          INC   HL            ;PUT HIGH ORDER BYTE OF ADDRESS AT HEAD
                248          LD    B,(HL)        ;OF QUEUE INTO REG B
                249          INC   HL            ;HL=NEW HEAD OF QUEUE ADDRESS
                250          PUSH  HL            ;SAVE NEW HEAD OF QUEUE ADDRESS
                251          SBC   HL,DE         ;HL=CURRENT LOCATION-END QUEUE-CARRY
                252          POP   HL            ;HL=ADD OF CURRENT TAIL OF QUEUE
                253          JP    M,B_#SYM      ;IF SUBTRACTION NEGATIVE JUMP
                254          LD    HL,#START     ;HL=ADDRESS OF START OF QUEUE
                255  B_#SYM  LD    (#HEAD),HL    ;STORE VALUE OF HL INTO TAIL ADDRESS
                256          ENDM
                257  ;////////////////////////////////////////////////////////////////
                258  ;******THIS CONCLUDES THE INITIALIZATION PART OF THE        ******
                259  ;OPERATING SYSTEM FOR PROCESSOR BOARD #1
                260  ;////////////////////////////////////////////////////////////////
                261  ;
                262  ;
                263  ;******THE NEXT SECTIONS BEGINS THE SECOND PART OF THE OPER-******
                264  ;ATING SYSTEM FOR PROCESSOR #1. THE OPERATING SYSTEM MONITORS
                265  ;THE TRANSMIT LOCAL QUEUE (LOTXHQ) AND THE TRANSMIT LOCAL BUSY
                266  ;QUEUE (LBTXHQ). ONCE A MESSAGE (MEMORY BLOCK ADDRESS) IS DE-
                267  ;TECTED IN THE TRANSMIT LOCAL QUEUE, IT IS REMOVED FROM THE
                268  ;QUEUE. THE ADDRESS OF THE MESSAGE IS DETERMINED THROUGH A
                269  ;LINER SEARCH OF A NETWORK ADDRESS TABLE (NWADTB). THE MESSAGE
                270  ;ADDRESS IS COMPARED WITH THE TABLE ENTRIES UNTIL A MATCH IS
                271  ;FOUND. THE LOCATION WITHIN THE NETWORK TABLE CORRESPONDS TO
                272  ;A LOCATION WITHIN A LOCAL ADDRESS TABLE (LOADTB). THIS LOCAL
                273  ;ADDRESS TABLE PROVIDES THE I/O ADDRESS FOR THE MESSAGE. NEXT
                274  ;THE MESSAGE CONTROL WORD IS TESTED TO DETERMINE IF THIS
                275  ;MESSAGE IS PART OF A MESSAGE SEQUENCE. THIS MESSAGE CONTROL
                276  ;WORD HAS BEEN ESTABLISHED TO HANDLE MESSAGES WHICH EXCEED
                277  ;THE MEMORY BLOCK STORAGE SIZE. IT IS AN EIGHT BIT WORD WHICH
                278  ;IS SENT AS THE LAST EIGHT BITS OF THE MESSAGE. THE WORD IS
                279  ;USED AS FOLLOWS:
                280  ;       BIT 0-2  MESSAGE SEQUENCE NUMBER
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND      COMMENT

                281  :       BIT 3    SET IF MESSAGE PART OF A SEQUENCE
                282  :       BIT 4    SET IF END MESSAGE OF SEQUENCE
                283  :       BIT 5-7  NOT USED
                284  :IF THE MESSAGE IS PART OF A SEQUENCE, MULTIBUFFER STORAGE
                285  :AREAS ARE SCANNED TO DETERMINE IF THE RECEIVED MESSAGE IS
                286  :A PART OF THE SEQUENCE. IF IT IS, THE BLOCK STORAGE ADDRESS
                287  :IS STORED IN THE MULTIBUFFER STORAGE AREA UNTIL THE COMPLETE
                288  :MESSAGE HAS ARRIVED. THESE MULTIBUFFER STORAGE AREA ARE
                289  :LOCATIONS WHERE THE MESSAGE'S BLOCK STORAGE ADDRESS CAN BE
                290  :STORED PLUS A STATUS WORD WHICH SIGNIFIES IF THE BUFFERS
                291  :ARE FULL (BIT 7 SET), NUMBER OF ADDRESSES IN THE MULTI-
                292  :BUFFER AND THE NUMBER NEEDED. IF THE MESSAGE IS NOT A PART
                293  :OF THE SEQUENCE, THE COMMAND WORD OF THE USART FOR THE
                294  :SPECIFIED I/O ADDRESS IS TESTED TO DETERMINE IF THE TRANS-
                295  :MITTER IS BUSY. IF IT IS THE MESSAGE IS PUT ON THE LOCAL
                296  :TRANSMIT BUSY QUEUE. IF NOT THE TRANSMITTER IS ENABLED
                297  :AND AN INTERRUPT INITIATED METHOD USED TO TRANSMIT THE
                298  :MESSAGE TO THE LOCAL SUBSCRIBER. THE OPERATING SYSTEM
                299  :MONITOR THE LOCAL TRANSMIT BUSY QUEUE RECHECKS TO SEE IF
                300  :THE TRANMITTER IS BUSY AND EITHER INITIATES TRANSMISSION
                301  :OR REPLACES THE MESSAGE BACK ON THE QUEUE. BIT 7 OF OF MULTI-
                302  :BUFFER STATUS WORDS ARE CHECKED TO DETERMINE WHEN A COMPLETE
                303  :SEQUENCE OF MESSAGES HAVE BEEN RECEIVED.
                304  :                 OTHER IMPORTANT LABELS
                305  : MBSAD1 -STATUS WORD MULTIBUFFER STORAGE AREA #1
                306  : MBSAD2 -STATUS WORD MULTIBUFFER STORAGE AREA #2
                307  : LOTXOR -STATUS WORD OF LOCAL TRANSMIT QUEUE
                308  :                       REGISTERS
                309  :THE OPERATING SYSTEM USES THE PRIMARY REGISTER SET THUS
                310  :THE SECONDARY SET ARE AVAILABLE FOR USE BY THOSE PROGRAMS
                311  :WHICH INTERRUPT THE MAIN OPERATING PROGRAM. THE MAIN OPERA-
                312  :TING DOES USE REGISTERS IX AND IY THUS THESE MUST BE SAVED ON
                313  :THE STACK IF USED BY OTHER SUBPROGRAMS
                314  :*****THIS SECTION CHECK THE STATUS WORD OF THE MULTIBUFFERS*****
                315  :TO DETERMINE IF THEY ARE COMPLETE. IF ONE IS, THE PROGRAM JUMP
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND      COMMENT

                316  :TO (MBTXCK) TO CHECK IF TRANSMITTER IS BUSY. IF NOT MBTXLO
                317  :IS EXECUTED TO LOAD THE BUFFER ADDRESSES IN THE USART TRANSMIT
                318  :STORAGE AREA AND TO ENABLE THE TRASMITTER. THE MULTIBUFFER
                319  :STORAGE AREA IS ORGANIZED IN THE FOLLOWING WAY
                320  :    MBSAXX   -STATUS WORD FOR THE BUFFER
                321  :           +1 -I/O ADDRESS OF USART HOLDING REGISTER
                322  :           +2 -ADDRESS OF LOCATION USED TO STORE THE MEMORY
                323  :           +3 -BLOCK ADDRESS OF MESSAGE BEING TRANSMIT LOCALLY
                324  :               EACH OF THE USART HAVE A LOCATION FOR THIS
                325  :           +4 -FROM ADDRESS OF MESSAGE
                326  :           +5 THROUGH +18 LOCATIONS FOR STORAGE OF THE DIFFER-
                327  :              ENT MEMORY BLOCK ADDRESSES OF THE SEQUENCED MSG
                328  :TO SUPPORT THE MESSAGE SEQUENCE OPERATION EIGHT BYTES ARE
                329  :RESERVED FOR EACH USART TO STORE THE FOLLOWING INFORMATION :
                330  :    TXURXX   -TX MSG LOW BYTE MEMORY BLOCK ADDRESS
                331  :           +1 -TX MSG HIGH BYTE MEMORY BLOCK ADDRESS
                332  :           +2 -MULTIBUFFER ADDESS OF NEXT MEMORY
                333  :           +3 -BLOCK ADDRESS
                334  :           +4 -ADDRESS OF MULTIBUFFER
                335  :           +5 -STATUS WORD
                336  :           +6 -NUMBER OF WORDS TRANSFERRED
                337  :           +7 -00000000
00F7 FB         338  MAIN     EI                  ;ENABLE INTERRUPT
00F8 211238     339  MAIN01   LD   HL,MBSA01      ;HL=MULTIBUFFER #1 STATUS WORD
00FB CB7E       340           BIT  7,(HL)         ;TEST IF MESSAGE ASSEMBLY COMPLETE
00FD C20801     341           JP   NZ,MBTXCK      ;IF SET JUMP
0100 212538     342           LD   HL,MBSA02      ;HL=MULTIBUFFER #2 STATUS WORD
0103 CB7E       343           BIT  7,(HL)         ;TEST IF MESSAGE ASSEMBLY COMPLETE
0105 CA4101     344           JP   Z,MAIN02       ;IF NOT SET JUMP
0108 E5         345  MBTXCK   PUSH HL             ;SAVE ADDRESS OF STATUS WORD
0109 23         346           INC  HL             ;HL=ADDRESS OF I/O PORT ADDRESS BYTE
010A 4E         347           LD   C,(HL)         ;C=I/O ADDRESS OF USART HOLDING REG
010B 0C         348           INC  C              ;SET C TO I/O ADDRESS OF COMMAND REG
010C 0C         349           INC  C
010D 0C         350           INC  C
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND      COMMENT

010E ED78       351          IN   A,(C)        ;A=COMMAND WORD
0110 CB47       352          BIT  0,A          ;TEST IF TRANSMITTER BUSY
0112 CA1901     353          JP   Z,MBTXLD     ;JUMP IF NOT BUSY
0115 E1         354          POP  HL           ;POP STATUS WORD
0116 C34101     355          JP   MAIN02       ;JUMP TO NEW SECTION
                356  ;***** THIS SECTION LOADS THE USART TX BUFFER POINTERS WITH ******
                357  ;THE START ADDRESS OF THE FIRST MESSAGE IN THE SEQUENCE. IT
                358  ;ALSO LOADS THE MULTI BUFFER ADDRESS OF THE NEXT MESSAGE
                359  ;IN THE SEQUENCE AND THE ADDRESS OF THE STATUS WORD IN
                360  ;LOCATIONS TXURXX+2 THROUGH +5. THIS SECTION IS ENTERED
                361  ;WITH THE STATUS WORD ON THE STACK AND HL=ADDRESS OF MULTI
                362  ;BUFFER STATUS WORD+1, C=COMMAND REGISTER ADDRESS A=CMD WORD
0119 23         363  MBTXLD  INC  HL           ; HL=STATUS WORD + 2
011A 5E         364          LD   E,(HL)       ;LOW BYTE ADD USART TX MEMORY POINTER
011B 23         365          INC  HL
011C 56         366          LD   D,(HL)       ;HIGH BYTE ADD USART TX MEMORY POINTER
011D 23         367          INC  HL
011E 23         368          INC  HL           ;STATUS WORD+5=LOW BYTE ADD MEMORY BLK
011F E5         369          PUSH HL           ;SAVE ADDRESS OF STATUS WORD + 5
0120 D5         370          PUSH DE           ;SAVE ADD OF USART TX MEMORY POINTER
0121 5E         371          LD   E,(HL)       ;DE=MEMORY BLOCK STORAGE
0122 23         372          INC  HL           ;   ADDRESS OF THE FIRST MESSAGE
0123 56         373          LD   D,(HL)       ;   IN THE SEQUENCE
0124 EB         374          EX   DE,HL        ;HL=DE
0125 D1         375          POP  DE           ;DE=USART TX MEMORY POINTER
0126 EDA0       376          LDI               ;PUT MEMORY BLOCK STORAGE ADDRESS
0128 EDA0       377          LDI               ;INTO USART TX MEMORY POINTER
012A E1         378          POP  HL           ;STATUS WORD+5=LOW BYTE ADD MEMORY BLK
012B 23         379          INC  HL           ;STATUS WORD+7=LOW BYTE ADD OF MEMORY
012C 23         380          INC  HL           ;BLOCK FOR 2ND PART OF MESSAGE SEQUENCE
012D 47         381          LD   B,A          ;B=COMMAND WORD
012E 7D         382          LD   A,L          ;A=LOW BYTE OF STATUS WORD + 7
012F 12         383          LD   (DE),A       ;STORE A AT TXURXX + 2
0130 13         384          INC  DE           ;DE=TXURXX + 3
0131 7C         385          LD   A,H          ;A=HIGH BYTE OF STATUS WORD + 7
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

0132 12         386           LD   (DE),A        ;STORE A AT LOCATION TXURXX + 3
0133 E1         387           POP  HL            ;HL=ADDRESS OF STATUS WORD
0134 13         388           INC  DE            ;DE=TXURXX + 4
0135 7D         389           LD   A,L           ;A=LOW BYTE OF STATUS WORD ADDRESS
0136 12         390           LD   (DE),A        ;STORE A AT TXURXX + 4
0137 13         391           INC  DE            ;DE=TXURXX + 5
0138 7C         392           LD   A,H           ;A=HIGH BYTE OF STATUS WORD ADDRESS
0139 12         393           LD   (DE),A        ;STORE A AT TXURXX + 5
013A 0C         394           INC  C             ;LDI INST DEC BC SO MUST INC C TWICE
013B 0C         395           INC  C             ;TO RESTORE THE I/O ADD OF COMMAND REG
013C 78         396           LD   A,B           ;A=USART COMMAND WORD
013D CB47       397           BIT  0,A           ;SET TX ENABLE BIT IN COMMAND WORD
013F ED79       398           OUT  (C),A         ;ENABLE THE TRANSMITTER
                399  :*****THIS SECTION CHECKS IF THERE IS A MESSAGE ON THE LOCAL*****
                400  :TRANSMIT QUEUE. IF THERE IS THE MEMORY BLOCK ADDRESS IS RE-
                401  :MOVED FROM THE QUEUE
0141 2A4B20     402  MAIN02  LD   HL,(LOTXHQ)  ;HL=HEAD OF QUEUE ADDRESS
0144 ED5B4D20   403           LD   DE,(LOTXTQ)  ;DE=TAIL OF QUEUE ADDRESS
0148 AF         404           XOR  A             ;A=0
0149 ED52       405           SBC  HL,DE         ;HL=HEAD ADDRESS - TAIL ADDRESS
014B CA7A01     406           JP   Z,MAIN03      ;IF ZERO JUMP
014E 214A20     407  DHLOG    LD   HL,LTXQFR     ;ADDRESS OF QUEUE STATUS WORD
0151 CBCE       408           SET  1,(HL)        ;STATUS WORD TO PROCESSOR #1 WAITING
0153 CB56       409           BIT  2,(HL)        ;CHECK IF PROCESSOR #2 WAITING
0155 C24E01     410           JP   NZ,DHLOG      ;JUMP IF PROCESSOR  IS WAITING
0158 CBC6       411           SET  0,(HL)        ;SET STATUS WORD TO PROCESSOR #1 USING
015A F3         412           DI                 ;DISABLE INTERRUPTS
015B                          SUBHQ LOTXSQ,LOTXEQ,LOTXSQ
015B 2A4F20     413 +         LD   HL,(LOTXSQ)   ;HL=ADDRESS OF HEAD OF QUEUE
015E 4E         413 +         LD   C,(HL)        ;PUT LOW ORDER BYTE OF ADDRESS AT
                413 +:                            HEAD OF QUEUE INTO REG C
015F 23         413 +         INC  HL            ;PUT HIGH ORDER BYTE OF ADDRESS AT HEAD
0160 46         413 +         LD   B,(HL)        ;OF QUEUE INTO REG B
0161 23         413 +         INC  HL            ;HL=NEW HEAD OF QUEUE ADDRESS
0162 E5         413 +         PUSH HL            ;SAVE NEW HEAD OF QUEUE ADDRESS
```

```
ADDR OBJECT    STMT  LABEL     OPCD OPERAND       COMMENT

0163 ED52       413 +          SBC  HL,DE         ;HL=CURRENT LOCATION-END QUEUE-CARRY
0165 E1         413 +          POP  HL            ;HL=ADD OF CURRENT TAIL OF QUEUE
0166 FA6C01     413 +          JP   M,B_0001      ;IF SUBTRACTION NEGATIVE JUMP
0169 214F20     413 +          LD   HL,LOTXSQ     ;HL=ADDRESS OF START OF QUEUE
016C 224F20     413 +B_0001    LD   (LOTXSQ),HL   ;STORE VALUE OF HL INTO TAIL ADDRESS
016F 214A20     414            LD   HL,LTXQFR
0172 CB86       415            RES  0,(HL)        ;SET STATUS WORD TO PROCESSOR #1
0174 CB8E       416            RES  1,(HL)        ;NOT WAITING OR USING
0176 FB         417            EI                 ;ENABLE THE INTERRUPTS
0177 C3C301     418            JP   MAIN04        ;
                419  ;****THIS SECTION CHECKS IF THERE IS A MESSAGE ON THE LOCAL******
                420  ;TRANSMIT BUSY QUEUE. IF THERE IS THE MEMORY BLOCK ADDRESS AND
                421  ;I/O PORT ADDRESS IS REMOVED FROM THE HEAD OF THE QUEUE
017A 2A4538     422  MAIN03    LD   HL,(LBTXHQ)   ;HL=HEAD OF QUEUE ADDRESS
017D ED5B4738   423            LD   DE,(LBTXTQ)   ;DE=TAIL OF QUEUE ADDRESS
0181 AF         424            XOR  A             ;A=0
0182 ED52       425            SBC  HL,DE         ;HL=HEAD ADDRESS - TAIL ADDRESS
0184 CAF800     426            JP   Z,MAIN01      ;IF ZERO JUMP TO START
0187 214538     427            LD   HL,LBTXHQ     ;HL=ADDRESS OF HEAD OF QUEUE
018A 5E         428            LD   E,(HL)        ;LOAD DE WITH
018B 23         429            INC  HL            ;THE LOCAL TABLE
018C 56         430            LD   D,(HL)        ;ADDRESS
018D 23         431            INC  HL            ;HL=NEW CURRENT ADDRESS
018E D5         432            PUSH DE            ;SAVE LOCAL TABLE ADDRESS
018F E5         433            PUSH HL            ;SAVE CURRENT LOCATION OF QUEUE
0190 114C39     434            LD   DE,LBTXEQ     ;DE=ADDRESS OF END OF QUEUE
0193 ED52       435            SBC  HL,DE         ;HL=CURRENT LOCATION-END OF QUEUE-CARRY
0195 FAA001     436            JP   M,MAIN3A      ;JUMP IF NEGATIVE
0198 E1         437            POP  HL            ;HL=INVALID ADDRESS
0199 214938     438            LD   HL,LBTXSQ     ;HL=ADDRESS OF START OF QUEUE
019C D1         439            POP  DE            ;DE=LOCAL TABLE ADDRESS
019D C3A201     440            JP   MAIN3B        ;JUMP OVER NEXT PART
01A0 E1         441  MAIN3A    POP  HL            ;HL=CURRENT LOCATION OF QUEUE ADDRESS
01A1 D1         442            POP  DE            ;DE=LOCAL TABLE ADDRESS
01A2 4E         443  MAIN3B    LD   C,(HL)        ;BC=MEMORY
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

01A3 23         444           INC  HL           ;BLOCK STORAGE
01A4 46         445           LD   B,(HL)       ;ADDRESS
01A5 C5         446           PUSH BC           ;PUT BLOCK STORAGE ADDRESS ON STACK
01A6 D5         447           PUSH DE           ;PUT LOCAL TABLE ADDRESS ON STACK
01A7 E5         448           PUSH HL           ;SAVE CURRENT LOCATION
01A8 114C39     449           LD   DE,L8TXEQ    ;DE=ADDRESS OF END OF QUEUE
01AB ED52       450           SBC  HL,DE        ;HL=CURRENT LOCATION-END OF QUEUE-CARRY
01AD F28801     451           JP   P,MAIN3C     ;JUMP IF POSITIVE
01B0 E1         452           POP  HL           ;HL=CURRENT LOCATION
01B1 224538     453           LD   (L8TXHD),HL  ;SET HEAD OF QUEUE TO CURRENT LOCATION
01B4 E1         454           POP  HL           ;HL=LOCAL TABLE ADDRESS
01B5 C3F501     455           JP   MAIN0E
01B8 E1         456  MAIN3C   POP  HL           ;HL=INVALID ADDRESS
01B9 214938     457           LD   HL,L8TXSQ    ;HL=ADDRESS OF START OF QUEUE
01BC 224538     458           LD   (L8TXHD),HL  ;SET HEAD OF QUEUE TO CURRENT LOCATION
01BF E1         459           POP  HL           ;HL=LOCAL TABLE ADDRESS
01C0 C3F501     460           JP   MAIN0E
                461  ;****THIS SECTION DETERMINES THE I/O ADDRESS OF THE RECEIVED*****
                462  ;MESSAGE THROUGH THE USE OF THE NETWORK ADDRESS TABLE (NWADTB)
                463  ;AND THE LOCAL ADDRESS TABLE (LOADTB). THE NETWORK ADDRESS
                464  ;TABLE CONSISTS OF ALL NETWORK ADDRESSES OF ALL THE SUB-
                465  ;SCRIBER CONNECTED TO THE NETWORK INTERFACE. FOR EACH ADDRESS
                466  ;IN THE NETWORK ADDRESS TABLE THE FOLLOWING IS INCLUDED IN THE
                467  ;LOCAL ADDRESS TABLE
                468  ;    I/O ADDRESS OF THE HOLDING REGISTER
                469  ;    LOW BYTE USART TX MEMORY POINTER
                470  ;    HIGH BYTE USART TX MEMORY POINTER
                471  ;ONCE THE NETWORK ADDRESS IS MATCHED THE LOCATION WITHIN THE
                472  ;LOCAL TABLE IS DETERMINE BY (2 * VALUE) + VALUE
01C3 C5         473  MAIN04   PUSH BC           ;SAVE MEMORY BLOCK ADDRESS
01C4 C5         474           PUSH BC           ;SAVE MEMORY BLOCK ADDRESS
01C5 DDE1       475           POP  IX           ;IX=MEMORY BLOCK ADDRESS
01C7 DD3500     476           DEC  (IX+0)       ;DECREASE MEMORY BLOCK STORAGE LENGTH
                477  ;                          BY ONE. THIS IS NEEDED SO THE
                478  ;                          MESSAGE CONTROL WORD IS NOT
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

               479   :                           TRANSMITTED TO A LOCAL PERIPHERAL
01CA DD7E02    480           LD   A,(IX+2)      :A=NETWORK ADDRESS OF MESSAGE
01CD 0E00      481           LD   C,0            :C=0
01CF 214720    482           LD   HL,NWADTB      :HL=ADDRESS OF NETWORK TABLE
01D2 BE        483   MAIN4A  CP   (HL)           :COMPARE A TO FIRST TABLE ENTRY
01D3 CAD801    484           JP   Z,MAIN4B       :IF EQUAL JUMP
01D6 0C        485           INC  C              :C=C + 1
01D7 23        486           INC  HL             :HL=NEXT ENTRY IN NETWORK TABLE
01D8 C3D201    487           JP   MAIN4A         :REPEAT THE LOOP UNTIL GET A MATCH
01DB 21C210    488   MAIN4B  LD   HL,LOADTB      :HL=ADDRESS OF LOCAL ADDRESS TABLE
01DE 79        489           LD   A,C            A=NETWORK TABLE OFFSET VALUE
01DF CB01      490           RLC  C              :C=2 * OFFSET VALUE
01E1 81        491           ADD  A,C            :A=2 * OFFSET VALUE + OFFSET VALUE
01E2 4F        492           LD   C,A            :C=A
01E3 0600      493           LD   B,0            :B=0
01E5 09        494           ADD  HL,BC          :HL=LOCAL ADDRESS TABLE OFFSET VALUE
               495   :*****THIS NEXT SECTION DETERMINES IF THE MESSAGE IS A PART******
               496   :OF A MESSAGE SEQUENCE. THIS IS DONE WITH THE MESSAGE CONTROL
               497   :WORD
01E6 DD4E00    498           LD   C,(IX+0)       :C=MESSAGE LENGTH
01E9 0600      499           LD   B,0            :B=0
01EB DD09      500           ADD  IX,BC          :IX=MEMORY BLOCK ADDRESS+MSG LENGTH+2
01ED DD7E01    501           LD   A,(IX+1)       :A=MESSAGE CONTROL WORD
01F0 CB67      502           BIT  4,A            :TEST BIT 4 OF CONTROL WORD
01F2 C21402    503           JP   NZ,MULT01      :IF BIT SET HAVE MULTIBUFFER MSG JUMP
01F5 4E        504   MAIN05  LD   C,(HL)         :C=I/O PORT ADDRESS
01F6 0C        505           INC  C              :
01F7 0C        506           INC  C              :
01F8 0C        507           INC  C              :C=I/O ADDRESS OF COMMAND REGISTER
01F9 ED78      508           IN   A,(C)          :A=COMMAND WORD
01FB CB47      509           BIT  0,A            :TEST IF TRANSMITTER IS ENABLE
01FD C20D02    510           JP   NZ,TXBUSY      :IF SET TRANSMITTER BUSY SO JUMP
0200 23        511           INC  HL             :E=LOW BYTE OF USART
0201 5E        512           LD   E,(HL)         :   TX MEMORY POINTER
0202 23        513           INC  HL             :D=HIGH BYTE OF USART
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

0203 56         514          LD   D,(HL)        :   TX MEMORY POINTER
0204 E1         515          POP  HL            :HL=MESSAGE'S MEMORY BLOCK STORAGE ADD
0205 7D         516          LD   A,L           :PUT THIS BLOCK
0206 12         517          LD   (DE),A        ;STORAGE ADDRESS
0207 13         518          INC  DE            :INTO USART
0208 7C         519          LD   A,H           ;TRANSMIT MEMORY
0209 12         520          LD   (DE),A        :POINTER
020A CBC7       521          SET  0,A           ;SET TRANSMITTER ENABLE IN COMMAND WORD
020C 0D         522          DEC  C             :SET C TO I/O ADDRESS
020D 0D         523          DEC  C             ;OF USART HOLDING
020E 0D         524          DEC  C             :REGISTER
020F ED79       525          OUT  (C),A         :OUTPUT COMMAND WORD
0211 C37A01     526          JP   MAIN03
                527  :****THIS SECTION DETERMINES IF THE RECEIVED MESSAGE IS PART*****
                528  :OF A SEQUENCE WHICH HAS ALREADY BEEN ALLOCATED A MULTI-
                529  :BUFFER ASSEMBLY STORAGE AREA. THIS IS DONE THROUGH A COMPAR-
                530  :ISON OF THE TO AND FROM ADDRESS OF THE MESSAGE. NOTE THIS
                531  :NECESSIATES THAT THE COMPLETE SEQUENCE ARRIVE AT THE INTER-
                532  :FACE PRIOR TO ANOTHER MESSAGE SEQUENCE WITH THE SAME TO/FROM
                533  :ADDRESSES. THIS LIMITATION CAN BE OVERCOME IF THE USER TO
                534  :USER PROTOCOL ESTABLISHES A TYPE OF MESSAGE NUMBERING IDENT-
                535  :IFICATION SYSTEM. THIS COULD THEN BE INCLUDED IN THIS SECTION
                536  :TO ALLOW DIFFERENTATION BETWEEN TWO MESSAGES WITH THE SAME TO/
                537  :FROM ADDRESSES WHICH REQUIRED SEQUENCING.UPON ENTERING THIS
                538  :SECTION THE STACK SHOULD CONTAINS THE MEMORY BLOCK ADDRESS
0214 E5         539  MULT01  PUSH HL            :SAVE LOCAL TABLE ADDRESS
0215 211238     540          LD   HL,MBSA01     :HL=ADDRESS OF STATUS WORD
0218 CB76       541          BIT  6,(HL)        :TEST IF AREA BEING USED
021A CA3B02     542          JP   Z,MULT1A      :IF AREA IS NOT IN USE JUMP
021D CB7E       543          BIT  7,(HL)        ;TEST IF ASSEMBLY HAS BEEN COMPLETED
021F C23B02     544          JP   NZ,MULT1A     ;ASSEMBLY COMPLETE SO JUMP
0222 3A1338     545          LD   A,(MBSA01+1)  :A=STATUS WORD + 1 = I/O PORT NUMBER
0225 E1         546          POP  HL            ;HL=LOCAL TABLE ADDRESS=I/O PORT NUMBER
0226 BE         547          CP   (HL)          :COMPARE THE TWO I/O PORT NUMBERS
0227 C23B02     548          JP   NZ,MULT1A     ;IF NOT EQUAL WRONG ASSEMBLY AREA
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

022A 3A1638     549           LD   A,(MBSA01+4) :A=STATUS WORD + 4 =FROM ADDRESS
022D FDE1       550           POP  IY           :IY=MEMORY BLOCK STORAGE ADDRESS
022F FDBE03     551           CP   (IY+3)       :COMPARE FROM ADDRESSES---NOTE THE OFF-
                552  :                           SET VALUE MUST BE CHANGED TO
                553  :                           CORRELATE TO THE MESSAGE
                554  :                           STRUCTURE IN USE---
0232 111238     555           LD   DE,MBSA01    :ADDRESS OF ASSEMBLY AREA STATUS WORD
0235 CA7902     556           JP   Z,MULT02     :IF MATCHES JUMP
0238 FDE5       557           PUSH IY           :SAVE MEMORY BLOCK ADDRESS
023A E5         558           PUSH HL           :SAVE LOCAL TABLE ADDRESS
023B 212538     559  MULT1A   LD   HL,MBSA02    :HL=ADDRESS OF STATUS WORD
023E CB76       560           BIT  6,(HL)       :TEST IF AREA BEING USED
0240 CAC702     561           JP   Z,MULT03     :IF AREA NOT IN USE JUMP
0243 CB7E       562           BIT  7,(HL)       :TEST IF ASSEMBLY HAS BEEN COMPLETED
0245 C2C702     563           JP   NZ,MULT03    :ASSEMBLY COMPLETE SO JUMP
0248 3A2638     564           LD   A,(MBSA02+1) :A=I/O PORT ADDRESS FROM ASSEMBLY AREA
024B E1         565           POP  HL           :LOCAL TABLE ADD OF ENTRY WHICH CONTAIN
024C BE         566           CP   (HL)         :I/O PORT ADDRESS OF RECEIVED MESSAGE
024D C2C702     567           JP   NZ,MULT03    :IF THEY DO NOT MATCH JUMP
0250 3A2938     568           LD   A,(MBSA02+4) :A=FROM ADDRESS FROM ASSEMBLY AREA
0253 DDE1       569           POP  IX           :IX=MESSAGE BLOCK ADDRESS
0255 FDBE03     570           CP   (IY+3)       :COMPARE FROM ADDRESSES---NOTE THE OFF-
                571  :                           SET VALUE MUST BE CHANGED TO
                572  :                           CORRELATE TO THE MESSAGE
                573  :                           STRUCTURE IN USE---
0258 112538     574           LD   DE,MBSA02    :DE=ADDRESS OF STATUS WORD
025B CA7902     575           JP   Z,MULT02     :IF MATCH JUMP
025E FDE5       576           PUSH IY           :SAVE MEMORY BLOCK ADDRESS
0260 E5         577           PUSH HL           :SAVE LOCAL TABLE ADDRESS
                578  :****SINCE THE RECEIVED MESSAGE ADDRESS DID NOT MATCH THE  ******
                579  :IN USE MULTIBUFFER ASSEMBLE AREA, THIS SECTION DETERMINES IF
                580  :A ASSEMBLY AREA IS AVAILABLE FOR USE. IF AN AREA IS NOT AVAIL-
                581  :ABLE THE MESSAGE IS PUT AT THE TAIL OF THE LOCAL TRANSMIT QUEUE.
                582  :UPON ENTERING THIS SECTION THE STACK SHOULD CONTAIN THE LOCAL
                583  :TABLE ADDRESS AND THE MEMORY BLOCK ADDRESS
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

0261 211238     584           LD   HL,MBSA01     :HL=ADDRESS OF STATUS WORD
0264 CB76       585           BIT  6,(HL)        :TEST IF ASSEMBLY AREA IN USE
0266 CAC702     586           JP   Z,MULT03      :IF NOT IN USE JUMP
0269 212538     587           LD   HL,MBSA02     :HL=ADDRESS OF STATUS WORD
026C CB76       588           BIT  6,(HL)        :TEST IF ASSEMBLY AREA IN USE
026E CAC702     589           JP   Z,MULT03      :IF NOT IN USE JUMP
0271 E1         590           POP  HL            :CLEAR THE STACK
0272 C1         591           POP  BC            :BC=MEMORY BLOCK STORAGE ADDRESS
0273 CD5C04     592           CALL LOTXQ         :PUT MEMORY BLOCK ADD AT END OF QUEUE
0276 C3F501     593           JP   MAINQF        ;JUMP TO MONITOR LOCAL TX BUSY QUEUE
                594  :****THIS SECTION IS EXECUTED IF THE RECEIVED MESSAGE HAS  ******
                595  :BEEN IDENTIFIED AS A PART OF A MESSAGE SEQUENCE WHICH HAS
                596  :BEEN ALLOCATED AN ASSEMBLY AREA. THE MESSAGE BLOCK STORAGE
                597  :ADDRESS IS STORED IN THE APPROPRIATE PLACE WITHIN THE ASSEMBLY
                598  :AREA. THE ASSEMBLY STATUS WORD IS UPDATED TO REFLECT ANOTHER
                599  :PART OF SEQUENCE HAS BEEN RECEIVED. A CHECK IS MADE TO DETER-
                600  :MINE IF THIS IS END PART OF THE SEQUENCE. IF THIS IS THE CASE
                601  :THE STATUS WORD IS CHANGED TO REFLECT THIS END SEQUENCE NUMBER
                602  :A TEST IS ALMOST MADE TO DETERMINE IF ALL PARTS OF MESSAGE
                603  :SEQUENCE HAVE BEEN RECEIVED. UPON ENTERING THIS SECTION THE
                604  :REGISTER CONTENTS ARE AS FOLLOWS
                605  :    IX -END OF MESSAGE ADDRESS
                606  :    DE -ADDRESS OF ASSEMBLY AREA
                607  :    HL -LOCAL TABLE ADDRESS POINTER
                608  :    IY -MEMORY BLOCK STORAGE ADDRESS
                609  :    SP -MEMORY BLOCK STORAGE ADDRESS
0279 E1         610  MULT02  POP  HL            :HL=MEMORY BLOCK STORAGE ADDRESS
027A D5         611           PUSH DE            :SAVE ADDRESS OF ASSEMBLY AREA
027B DD7E01     612           LD   A,(IX+1)      :A=CONTROL WORD OF MESSAGE
027E E66F       613           AND  00000111      :A=MESSAGE SEQUENCE NUMBER
0280 13         614           INC  DE
0281 13         615           INC  DE
0282 13         616           INC  DE
0283 13         617           INC  DE
0284 13         618           INC  DE            :DE=STATUS WORD ADDRESS + 5
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND      COMMENT

0285 3D         619  MULT2B   DEC  A            ;SET UP COUNTER TO DETERMINE PROPER
                620  ;                          ASSEMBLY AREA STORAGE ADDRESS
                621  ;                          FOR THE MEMORY BLOCK ADDRESS
0286 FA8E02     622           JP   M,MULT2A     ;IF MINUS JUMP OUT OF LOOP
0289 13         623           INC  DE           ;INCREMENT THE ADDRESS BY TWO
028A 13         624           INC  DE           ;
028B C38502     625           JP   MULT2B       ;REPEAT THE LOOP
028E 7D         626  MULT2A   LD   A,L          ;STORE THE MEMORY
028F 12         627           LD   (DE),A       ;BLOCK ADDRESS AT
0290 13         628           INC  DE           ;APPROPRIATE LOCATION
0291 7C         629           LD   A,H          ;OF THE ASSEMBLY
0292 12         630           LD   (DE),A       ;AREA
0293 DD7E01     631           LD   A,(IX+1)     ;A=CONTROL WORD OF MESSAGE
0296 E1         632           POP  HL           ;HL=ADDRESS OF STATUS WORD
0297 CB6F       633           BIT  5,A          ;TEST IF END OF SEQUENCE MESSAGE
0299 C2B302     634           JP   NZ,MULT2C    ;IF END OF SEQUENCE JUMP
029C 34         635  MULT2D   INC  (HL)         ;INCREMENT THE STATUS WORD TO REFLECT
                636  ;                          MESSAGE JUST RECEIVED
029D 7E         637           LD   A,(HL)       ;A=NEW STATUS WORD
029E CBB7       638           RES  6,A          ;SET BIT SIX OF STATUS WORD TO ZERO.
                639  ;                          THIS BIT IS SET TO ONE IN STATUS
                640  ;                          WORD SINCE ASSEMBLY AREA IS IN
                641  ;                          USE. IT MUST BE SET TO ZERO TO
                642  ;                          ALLOW COMPARISON OF BITS 0-2
                643  ;                          WITH BITS 3-5
02A0 47         644           LD   B,A          ;B=MODIFIED STATUS WORD
02A1 1F         645           RRA               ;SHIFT BITS 3-5 OF MODIFIED STATUS
02A2 1F         646           RRA               ;WORD INTO BIT POSITION 0-2. THESE BITS
02A3 1F         647           RRA               ;;REFLECT END OF SEQUENCE NUMBER
02A4 CB98       648           RES  3,B          ;SET BIT 3 THROUGH 5 TO ZERO. REGISTER
02A6 CBA0       649           RES  4,B          ;B NOW CONTAINS THE NUMBER OF MESSAGES
02A8 CBA8       650           RES  5,B          ;IN SEQUENCE WHICH HAVE BEEN RECEIVED
02AA A6         651           XOR  B            ;A EXCLUSE OR WITH B
02AB C24101     652           JP   NZ,MAIN02    JUMP NOT ZERO TO LOCAL TX QUEUE MONITOR
02AE CBFE       653           SET  7,(HL)       ;SET STATUS WORD TO ASSEMBLY COMPLETE
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

02B0 C3F800     654           JP   MAIN01        ;JUMP TO START OF OPERATING SYSTEM
02B3 E66F       655  MULT2C   AND  00000111      ;A=END OF SEQUENCE NUMBER
02B5 CB07       656           RLC  A             ;SHIFT END OF SEQUENCE NUMBER
02B7 CB07       657           RLC  A             ;TO BITS 3 THROUGH 5 OF
02B9 CB07       658           RLC  A             ;REGISTER A
02BB 46         659           LD   B,(HL)        ;B=CURRENT ASSEMBLY STATUS WORD
02BC CB98       660           RES  3,B           ;SET BITS 3 THROUGH
02BE CBA0       661           RES  4,B           ;5 OF CURRENT STATUS
02C0 CBA8       662           RES  5,B           ;WORD TO ZERO
02C2 A8         663           XOR  B             ;UPDATE THE ASSEMBLY AREA STATUS WORD
02C3 77         664           LD   (HL),A        ;WITH END OF SEQUENCE NUMBER
02C4 C39C02     665           JP   MULT2D        ;JUMP TO CHECK IF ALL SEQUENCES RECEIVE
                666  ;****THIS SECTION IS EXECUTED IF THE RECEIVED SEQUENCE MESSAGE
                667  ;IS NOT PART OF THOSE IN THE ASSEMBLY AREAS AND AN ASSEMBLY
                668  ;AREA IS AVAILABLE FOR USE. THIS SECTION SET THE ASSEMBLY
                669  ;STATUS TO IN USE, LOADS THE TO/FROM ADDRESS OF THE MESSAGE IN
                670  ;THE ASSEMBLY AREA. UPON ENTERING THIS SECTION THE REGISTER
                671  ;CONTENTS ARE AS FOLLOWS:
                672  ;     IX  -END OF MESSAGE ADDRESS
                673  ;     IY  -MEMORY BLOCK ADDRESS
                674  ;     HL  -ADDRESS OF ASSEMBLY AREA STATUS WORD
                675  ;     SP  -LOCAL TABLE ADDRESS POINTER
                676  ;          MEMORY BLOCK STORAGE ADDRESS
02C7 CBF6       677  MULT03   SET  6,(HL)        ;SET ASSEMBLY STATUS WORD TO IN USE
02C9 EB         678           EX   DE,HL         ;DE=ADDRESS OF STATUS WORD
02CA E1         679           POP  HL            ;HL=LOCAL TABLE ADDRESS
02CB E5         680           PUSH HL            ;SAVE LOCAL TABLE ADDRESS
02CC D5         681           PUSH DE            ;SAVE STATUS WORD ADDRESS
02CD 13         682           INC  DE
02CE EDA0       683           LDI                ;LOAD I/O ADDRESS AND USART TX
02D0 EDA0       684           LDI                ;MEMORY POINTER ADDRESS INTO THE
02D2 EDA0       685           LDI                ;ASSEMBLY AREA
02D4 FD7E03     686           LD   A,(IY+3)      ;A=FROM ADDRESS OF MESSAGE---NOTE THIS
                687  ;                           OFF-SET VALUE MUST BE CHANGED
                688  ;                           TO CORRELATE TO THE MESSAGE
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND      COMMENT

               689   :                          STRUCTURE IN USE
02D7 12        690           LD   (DE),A        ;LOAD FROM ADDRESS INTO ASSEMBLY AREA
02D8 E1        691           POP  HL            ;HL=LOCAL TABLE ADDRESS
02D9 D1        692           POP  DE            ;DE=STATUS WORD ADDRESS
02DA C37902    693           JP   MULT02        ;
               694   ;****THIS SECTION IS EXECUTED IF THE USART TRANSMITTER     ******
               695   ;WAS BUSY TRANSMITTING A PREVIOUS MESSAGE. THIS SECTION PUTS
               696   ;THE MESSAGE ON THE TRANSMIT LOCAL BUSY QUEUE. THE FOLLOWING
               697   ;INFORMATION IS PUT ON THE QUEUE IN THE ORDER SPECIFIED
               698   ;       LOW BYTE LOCAL TABLE ADDRESS
               699   ;       HIGH BYTE LOCAL TABLE ADDRESS
               700   ;       LOW BYTE MEMORY BLOCK ADDRESS
               701   ;       HIGH BYTE MEMORY BLOCK ADDRESS
               702   ;UPON ENTERING THIS SECTION THE REGISTERS CONTENTS ARE AS FOLLOWS
               703   ;       HL - LOCAL TABLE ADDRESS
               704   ;       SP - MEMORY BLOCK STORAGE ADDRESS
02DD E5        705   TXBUSY  PUSH HL            ;SAVE LOCAL TABLE ADDRESS
02DE 2A4738    706           LD   HL,(LBTXTQ)   ;HL=ADDRESS OF TAIL OF QUEUE
02E1 D1        707           POP  DE            ;DE=LOCAL TABLE ADDRESS
02E2 73        708           LD   (HL),E        ;STORE THE LOCAL TABLE
02E3 23        709           INC  HL            ;ADDRESS AT THE TAIL OF
02E4 72        710           LD   (HL),D        ;THE QUEUE
02E5 23        711           INC  HL            ;INCREMENT AND SAVE
02E6 E5        712           PUSH HL            ;NEW CURRENT LOCATION
02E7 114C39    713           LD   DE,LBTXEQ     ;DE=ADDRESS OF END OF QUEUE
02EA ED52      714           SBC  HL,DE         ;HL=CURRENT LOCATION-END OF QUEUE-CARRY
02EC FAF602    715           JP   M,TXBUS1      ;SKIP NEXT PART
02EF E1        716           POP  HL            ;HL=INVALID ADDRESS
02F0 214938    717           LD   HL,LBTXSQ     ;HL=ADDRESS OF START OF QUEUE
02F3 C3F702    718           JP   TXBUS2
02F6 E1        719   TXBUS1  POP  HL            ;HL=VALID CURRENT LOCATION ADDRESS
02F7 D1        720   TXBUS2  POP  DE            ;DE=MEMORY BLOCK STORAGE ADDRESS
02F8 73        721           LD   (HL),E        ;STORE THE MEMORY
02F9 23        722           INC  HL            ;BLOCK ADDRESS AT THE
02FA 72        723           LD   (HL),D        ;TAIL OF THE QUEUE
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

02FB 23         724          INC  HL            ;
02FC E5         725          PUSH HL            ;SAVE THE CURRENT LOCATION ADDRESS
02FD 114C39     726          LD   DE,LBTXED     ;DE=ADDRESS OF END OF QUEUE
0300 ED52       727          SBC  HL,DE         ;HL=CURRENT LOCATION-END OF QUEUE-CARRY
0302 F20C03     728          JP   P,TXBUS3      ;IF POSITIVE JUMP
0305 E1         729          POP  HL            ;HL=CURRENT LOCATION
0306 224738     730          LD   (LBTXTD),HL   ;TAIL OF QUEUE = CURRENT LOCATION
0309 C34101     731          JP   MAIN02        ;JUMP TO LOCAL TX QUEUE MONITOR
030C E1         732  TXBUS3  POP  HL            ;HL=INVALID ADDRESS
030D 214938     733          LD   HL,LBTXSD     ;HL=ADDRESS OF START OF QUEUE
0310 224738     734          LD   (LBTXTD),HL   ;TAIL OF QUEUE = CURRENT LOCATION
0313 C34101     735          JP   MAIN02        ;JUMP TO LOCAL TX QUEUE MONITOR
                736  ;*************************************************************
                737  ;*************   SUBROUTINE INITIALIZATION      ***************
                738  ;*************************************************************
                739  ;THIS SUBROUTINE OUTPUTS INITIALIZATION WORDS TO THE VARIOUS
                740  ;CHIPS ASSOCIATED WITH PROCESSOR BOARD #1 AND THE INPUT CARD. THE
                741  ;ROUTINE REQUIRES THE INFORMATION BE IN THE FORMAT OF THE NON
                742  ;SYNCHRONOUS USART PARAMETER LIST DESCRIBED PREVIOUSLY. UPON
                743  ;ENTRY THE HL REGISTER MUST CONTAIN THE ADDRESS OF THE FIRST
                744  ;ENTRY IN THE PARAMETER LIST. UPON EXIT THE BC REGISTER
                745  ;CONTAINS ADDRESS OF NEXT PARAMETER TABLE IN THE LINKED LIST
0316 3E03       746  ITUART  LD   A,3           ;A=3
0318 86         747          ADD  A,(HL)        ;A=I/O ADDRESS OF HOLDING REGISTER + 3
0319 4F         748          LD   C,A           ;C=A=ADDRESS OF COMMAND REGISTER
031A 23         749          INC  HL            ;
031B 23         750          INC  HL            ;
031C 23         751          INC  HL            ;HL=START OF LIST ADDRESS + 3
031D EDA3       752          OUTI               ;OUTPUT COMMAND WORD
031F 0D         753          DEC  C             ;C=ADDRESS OF MODE REGISTERS
0320 EDA3       754          OUTI               ;OUTPUT MODE REGISTER #1 WORD
0322 EDA3       755          OUTI               ;OUTPUT MODE REGISTER #2 WORD
0324 4E         756          LD   C,(HL)        ;C=Z80A-CTC CHANNEL I/O ADDRESS
0325 23         757          INC  HL
0326 EDA3       758          OUTI               ;OUTPUT Z80A-CTC MODE WORD
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

0328 EDA3       759           OUTI               ;OUTPUT Z80A-CTC TIMER PRESCALER
032A 4E         760           LD   C,(HL)        ;C=LOW BYTE OF ADD NEXT PARAMETER LIST
032B 23         761           INC  HL            ;
032C 46         762           LD   B,(HL)        ;B=HIGH BYTE OF ADD NEXT PARAMETER LIST
032D C9         763           RET                ;RETURN
                764  ;*****************************************************************
                765  ;***************     SUBROUTINE USART TRANSMIT     ***************
                766  ;*****************************************************************
                767  ;THIS SUBROUTINE REPRESENTS A TYPICAL INTERRUPT SERVICE ROUTINE
                768  ;FOR TRANSMISSION OF A CHARACTER TO AN ASYNCHRONOUS TTY OR CRT.
                769  ;THE ROUTINE IMPLEMENTS THE MESSAGE SEQUENCING FEATURE OF THE
                770  ;INTERFACE THROUGH THE USE OF THE MESSAGE CONTROL WORD. UPON
                771  ;GENERATION OF A CHARACTER NEEDED INITIATED INTERRUPT, THE
                772  ;SERVICE ROUTINE CALCULATES THE STORAGE ADDRESS AND OUTPUTS
                773  ;THE NEXT WORD TO THE USART. THE ROUTINE THEN CHECKS TO
                774  ;DETERMINE IF THE WORD IS THE END OF MESSAGE. IF IT IS THE
                775  ;ROUTINE CHECKS IF THE MESSAGE WAS A PART OF A SEQUENCE OF
                776  ;MESSAGES. IF NOT THE TRANSMITTER IS DISABLED. IF IT WAS A
                777  ;PART OF THE SEQUENCE THE USART TX MEMORY POINTER IS UPDATED
                778  ;WITH NEXT BLOCK ADDRESS OF THE SEQUENCE. TO SUPPORT THIS
                779  ;ROUTINE EIGHT EIGHT BIT WORDS MUST BE ALLOCATED FOR USE BY
                780  ;THIS ROUTINE. THEY ARE AS FOLLOWS:
                781  ;    TXURXX    -TX MSG LOW BYTE MEMORY BLOCK ADDRESS
                782  ;          +1 -TX MSG HIGH BYTE MEMORY BLOCK ADDRESS
                783  ;          +2 -MULTIBUFFER ADDESS OF NEXT MEMORY
                784  ;          +3 -BLOCK ADDRESS
                785  ;          +4 -ADDRESS OF MULTIBUFFER
                786  ;          +5 -STATUS WORD
                787  ;          +6 -NUMBER OF WORDS TRANSFERRED
                788  ;          +7 -00000000
                789  ;THE I/O PORT ADDRESSES USED ARE ZERO THROUGH FOUR
032E 08         790  SATX01  EX   AF,AF' ;SAVE THE REGISTERS OF
032F D9         791           EXX                ;THE INTERRUPTED PROGRAM
0330 ED4B3D38   792           LD   BC,(TXJR01)   ;BC=MEMORY BLOCK STORAGE ADDRESS
0334 214338     793           LD   HL,TXUR01+6   ;HL=NUMBER OF WORDS TRANSFERRED
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND        COMMENT

0337 34          794          INC  (HL)           ;NUMBER OF WORDS TRANSFERRED + 1
0338 2A4338      795          LD   HL,(TXUR01+6)  ;HL=NUMBER OF WORDS TRANSFERRED
033B E5          796          PUSH HL             ;SAVE NUMBER OF WORDS TRANSFERRED
033C 09          797          ADD  HL,BC          ;HL=LOCATION OF NEXT WORD TO BE SENT-1
033D 23          798          INC  HL             ;ADJUST FOR +1 BLOCK STORAGE SITUATION
033E 7E          799          LD   A,(HL)         ;A=WORD TO BE TRANSMITTED
033F D300        800          OUT  (0),A          ;OUTPUT A TO USART HOLDING REGISTER
0341 0A          801          LD   A,(BC)         ;A=MESSAGE LENGTH
0342 ED5B4338    802          LD   DE,(TXUR01+6)  ;E=NUMBER OF WORDS TRANSFERRED
0346 BB          803          CP   E              ;COMPARE A WITH E
0347 CA5203      804          JP   Z,TXED01       ;IF ZERO END OF MESSAGE SO JUMP
034A 3E08        805  TXRT01  LD   A,00001000B   ; A=MASK WORD OF PIC
034C D319        806          OUT  (25D),A        ; RESET PIC
034E 08          807          EX   AF,AF'  ;RESTORE THE INTERRUPTED
034F D9          808          EXX                 ;PROGRAM REGISTERS
0350 ED4D        809          RETI                ;RETURN FROM INTERRUPT
0352 23          810  TXED01  INC  HL             ;HL=ADDRESS OF MESSAGE CONTROL WORD
0353 CB66        811          BIT  4,(HL)         ;DETERMINE IF MULTIBUFFER MESSAGE
0355 C26903      812          JP   NZ,TXED1A      ;IF MULTIBUFFER MESSAGE JUMP
0358 DB03        813  TXED1B  IN   A,(3)          ;INPUT COMMAND WORD
035A CB47        814          BIT  0,A            ;RESET TRANSMITTER ENABLE BIT
035C D303        815          OUT  (3),A          ;DISABLE THE TRANSMITTER
035E 3E00        816          LD   A,0            ;A=0
0360 324338      817          LD   (TXUR01+6),A   ;NUMBER OF WORDS TRANSFERRED = 0
0363 CDC304      818          CALL DALMEN         ;RETURN THE MEMORY BLOCK
0366 C34A03      819          JP   TXRT01         ;JUMP TO RETURN SECTION
0369 23          820  TXED1A  INC  HL             ;HL=ADDRESS OF CONTROL WORD
036A CB6E        821          BIT  5,(HL)         ;DETERMINE IF END OF MULTI BUFFER MSG
036C C28303      822          JP   NZ,TXED1C      ;IF IT IS JUMP
036F 2A3F38      823          LD   HL,(TXUR01+2)  ;HL=ASSEMBLY AREA ADDRESS POINTER
0372 5E          824          LD   E,(HL)         ;LOAD THE NEXT MULTIBUFFER
0373 23          825          INC  HL             ;MEMORY BLOCK ADDRESS INTO
0374 56          826          LD   D,(HL)         ;REGISTER DE
0375 23          827          INC  HL             ;HL=NEW ASSEMBLY AREA ADDRESS POINTER
0376 223F38      828          LD   (TXUR01+2),HL  ;STORE NEW POINTER ADDRESS
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

0379 ED533D38   829           LD   (TXUR01),DE  :USART TX MEMORY POINTER = DE
037D CDC304     830           CALL DALMEN       :RETURN PREVIOUS BLOCK
0380 C34A03     831           JP   TXRT01       ;JUMP TO RETURN SECTION
0383 2A4138     832  TXED1C   LD   HL,(TXJR01+4) HL=ASSEMBLY AREA STATUS WORD ADDRESS
0386 3E37       833           LD   A,00110111B
0388 77         834           LD   (HL),A       :SET STATUS WORD TO NOT USED, NOT READY
                835  :                           NUMBER OF MSG RECEIVED=7 AND
                836  :                           END MESSAGE SEQUENCE NUMBER=6
0389 C35803     837           JP   TXED1B
                838  :*****************************************************
                839  :***************     SUBROUTINE USART RECEIVE     ***************
                840  :*****************************************************
                841  :THIS SUBROUTINE REPRESENTS A TYPICAL INTERRUPT SERVICE ROUTINE
                842  :FOR RECEIPT OF CHARACTERS FROM AN ASYNCHRONOUS TTY OR CRT. THE
                843  :ROUTINE IMPLEMENTS THE MESSAGE SEQUENCING FEATURE OF THE
                844  :INTERFACE THROUGH THE USE OF THE MESSAGE CONTROL WORD. UPON
                845  :GENERATION OF A CHARACTER INITIATED INTERRUPT, THE SERVICE
                846  :ROUTINE FIRST CHECKS TO DETERMINE IF MEMORY HAS BEEN ALLOCATED
                847  :FOR THE MESSAGE. IF NOT A MEMORY BLOCK IS OBTAINED, AND THE
                848  :CHARACTER STORED. SUBSEQUENT CHARACTERS ARE STORED AFTER
                849  :CHECKING FOR A USER PROGRAMMED DELETE OR END OF MESSAGE
                850  :CHARACTER. IN ADDITION, A CHECK IS MADE TO DETERMINE IF
                851  :THE MESSAGE EXCEEDS THE USER SPECIFIED MEMORY BLOCK SIZE. IF
                852  :IT DOES, THE APPROPRIATE CONTROL WORD IS ADDED AT THE END
                853  :OF THE MESSAGE AND THE MESSAGE PUT ONTO THE NETWORK TRANS-
                854  :MIT QUEUE. TO SUPPORT THIS ROUTINE, FIVE EIGHT BIT WORDS
                855  :MUST BE ALLOCATED FOR USE BY THIS ROUTINE. THEY ARE AS FOLLOWS
                856  :     RXURXX    -MEMORY BLOCK STORAGE ADDRESS
                857  :           +1 -FOR MESSAGE STORAGE
                858  :           +2 -CURRENT MESSAGE LENGTH
                859  :           +3  00000000
                860  :           +4 -MESSAGE CONTROL WORD
                861  :                        REGISTERS
                862  :THE SUBROUTINE USES THE ALTERNATE REGISTER SET. IT DOES NOT
                863  :REENABLE INTERRUPTS AND THUS CANNOT ITSELF BE INTERRUPTED
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND      COMMENT

               864   :                          SPECIAL INST.
               865   :SPECIAL CONSIDERATIONS ARE REQUIRED WHEN USING BOTH THE
               866   :DELETE CAPABILITY AND THE MESSAGE SEQUENCING CAPABILITY. IF
               867   :MORE THAN THE PREVIOUS CHARACTER IS DELETED IT IS POSSIBLE
               868   :TO TRY AND DELETE A CHARACTER FROM A MESSAGE WHICH IS AL-
               869   :READY ON THE TRANSMIT QUEUE. TO PREVENT THIS, DELETIONS
               870   :SHOULD BE LIMITED TO ONE CHARACTER.
038C 08        871   SARX01  EX    AF,AF' :SAVE THE REGISTERS OF
038D D9        872           EXX           :THE INTERRUPTED PROGRAM
038E ED4B3838  873           LD    BC,(RXUR01)  :BC=MEMORY BLOCK STORAGE ADDRESS
0392 AF        874           XOR   A            ;A=0
0393 80        875           ADD   A,B          ;A=0+HIGH BYTE BLOCK STORAGE ADDRESS
0394 DB00      876           IN    A,(0)        ;INPUT TO A FROM I/O PORT ADDRESS
               877   :                           ---NOTE THIS ADDRESS MUST
               878   :                           CORRELATE WITH USART PARAMETER
               879   :                           PARAMETER LIST
               880   :DO ANY CODE CONVERSION
               881   :THAT IS NECESSARY
0396 CAC103    882           JP    Z,NOMN01     :IF ZERO NEED MEMORY SO JUMP
0399 FE3F      883           CP    3FH          :COMPARE WITH SPECIFIED DELETE CHAR
               884   :                           ---DELETE CHA IS USER DEFINED---
039B CAD403    885           JP    Z,DELF01     :IF SAME CHARACTER THAN JUMP
039E FE24      886           CP    '$'          :COMPARE WITH SPECIFIED END OF MSG CHAR
               887   :                           ---EOM CHAR IS USER DEFINED---
03A0 CADB03    888           JP    Z,EOM01
03A3 213A38    889           LD    HL,RXJR01+2  :HL=ADDRESS OF LENGTH OF MESSAGE
03A6 34        890           INC   (HL)         ;INCREMENT THE MESSAGE LENGTH
03A7 ED5B3A38  891           LD    DE,(RXJR01+2) :DE=NEW MESSAGE LENGTH
03AB D5        892           PUSH  DE           ;SAVE NEW MESSAGE LENGTH
03AC 2A0011    893           LD    HL,(BLKSIZ)  :HL=MEMORY BLOCK SIZE
03AF 37        894           SCF                ;CARRY FLAG = 1
03B0 ED52      895           SBC   HL,DE        ;HL=BLOCK SIZE - CURRENT LENGTH -1
03B2 CAFD03    896           JP    Z,EXMN01     ;IF ZERO WILL EXCEED MEMORY NEXT TIME
               897   :                           THUS MUST STORE THIS CHARACTER
               898   :                           IN NEW MEMORY BLOCK AND STORE
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

                899  :                          THE MESSAGE CONTROL WORD AT
                900  :                          CURRENT LOCATION
0385 E1         901          POP  HL            ;HL=NEW MESSAGE LENGTH
0386 09         902  GTMN01  ADD  HL,BC         ;HL=BLOCK STORAGE ADD + NEW MSG LENGTH
0387 77         903          LD   (HL),A        ;STORE THE INPUTTED WORD
0388 3E08       904  SPOG01  LD   A,00001000B  ;LOAD A WITH PRIORITY CONTROLLER MASK
                905  :                          WORD. ---NOTE THIS MASK WORD
                906  :                          IS USER SPECIFIED---
038A D318       907          OUT  (240),A       ;OUTPUT MASK WORD TO I/O ADDRESS OF
                908  :                          PRIORITY CONTROLLER THUS REACT-
                909  :                          TIVATING THE CONTROLLER
                910  :                          ---NOTE THIS ADDRESS MUST COR-
                911  :                          RELATE WITH INPUT BOARD---
038C 08         912          EX   AF,AF' ;RESTORE THE INTERRUPTED
038D D9         913          EXX                ;PROGRAM REGISTERS
038E FB         914          EI                 ;ENABLE INTERRUPT
038F ED4D       915          RETI               ;RETURN FROM INTERRUPT
0391 F5         916  NOMN01  PUSH AF            ;SAVE THE RECEIVED CHARACTER
0392 CD8604     917          CALL ALLMEN        ;ALLOCATE A MEMORY BLOCK
0395 ED433838   918          LD   (RXUR01),BC   ;LOAD NEW BLOCK ADDRESS INTO USART
                919  :                          RX MEMORY POINTER
0399 AF         920          XOR  A             ;A=0
039A 323C38     921          LD   (RXUR01+4),A ;SET CONTROL WORD TO ZERO
039D 210200     922          LD   HL,02         ;HL=2
03A0 F1         923          POP  AF            ;RESTORE THE RECEIVED CHARACTER
03A1 C38603     924          JP   GTMN01
03A4 213A38     925  DELE01  LD   HL,RXJR01+2  ;HL=MESSAGE LENGTH ADDRESS
03A7 35         926          DEC  (HL)          ;DECREMENT MESSAGE LENGTH
03A8 C38803     927          JP   SPOG01        ;JUMP TO RETURN SECTION
03AB 213A38     928  EOM01   LD   HL,RXUR01+2  ;HL=MESSAGE LENGTH ADDRESS
03AE 34         929          INC  (HL)          ;INCREMENT MESSAGE LENGTH
03AF 2A3A38     930          LD   HL,(RXUR01+2) ;HL=NEW MESSAGE LENGTH
03B2 09         931          ADD  HL,BC         ;HL=BLOCK STORAGE ADDRESS + MSG LENGTH
03B3 113C38     932          LD   DE,RXJR01+4   ;DE=CONTROL WORD ADDRESS
03B6 1A         933          LD   A,(DE)        ;A=CONTROL WORD
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

03E7 CBEF       934          SET  5,A           ;SET END OF MESSAGE BIT
03E9 77         935          LD   (HL),A        ;STORE CONTROL WORD AT END OF MESSAGE
03EA 3A3A38     936          LD   A,(RXJR01+2)  ;A=MESSAGE _ENGTH
03ED 02         937          LD   (BC),A        ;STORE LENGTH AT START OF MEMORY BLOCK
03EE CD3204     938          CALL NWTXQ         ;PUT MESSAGE ON NETWORK TX QUEUE
03F1 AF         939          XOR  A             ;A=0
03F2 323938     940          LD   (RXUR01+1),A  ;SET HIGH BYTE OF USART RX PTN = ZERO
03F5 3E01       941          LD   A,1           ;A=1
03F7 323A38     942          LD   (RXUR01+2),A  ;SET MESSAGE LENGTH = 1
03FA C33803     943          JP   SPOG01        ;JUMP TO RETURN SECTION
03FD E1         944  EXMN01  POP  HL            ;HL=INCREMENTED MESSAGE LENGTH
03FE 09         945          ADD  HL,BC         ;HL=BLOCK ADDRESS + NEW MSG LENGTH
03FF EB         946          EX   DE,HL         ;DE=STORAGE ADDRESS
0400 213C38     947          LD   HL,RXJR01+4   ;HL=CONTROL WORD ADDRESS
0403 CBE6       948          SET  4,(HL)        ;SET MESSAGE SEQUENCE BIT
0405 EDA8       949          LDD                ;STORE CONTROL WORD AT THE STORAGE ADD
0407 CD3204     950          CALL NWTXQ         ;PUT MESSAGE ON NETWORK TRANSMIT QUEUE
040A C5         951          PUSH BC            ;SAVE PREVIOUS BLOCK STORAGE ADDRESS
040B F5         952          PUSH AF            ;SAVE THE INPUTTED CHARACTER
040C CD8604     953          CALL ALLMEN        ;GET NEW BLOCK STORAGE ADDRESS
040F ED433838   954          LD   (RXUR01),BC   ;STORE BLOCK ADD IN USART RX MEMORY PNT
0413 210200     955          LD   HL,2          ;HL=2
0416 09         956          ADD  HL,BC         ;HL=ADDRESS OF FIRST AVAILABLE STORAGE
0417 D1         957          POP  DE            ;DE=PREVIOUS BLOCK STORAGE ADDRESS
0418 EB         958          EX   DE,HL         ;HL=PREVIOUS BLOCK STORAGE ADDRESS
                959  :                          DE=FIRST AVAILABLE STORAGE
                960  :                          BLOCK ADDRESS
0419 23         961          INC  HL            ;HL=ADDRESS OF START OF INFORMATION
041A 23         962          INC  HL            ;FROM PREVIOUS MESSAGE
041B 010200     963          LD   BC,2          ;BC=NUMBER OF WORDS TO BE DUPLICATED
041E 3E03       964          LD   A,3           ;A=NUMBER OF WORDS TO BE DUPLICATED + 1
                965  :                          ---THE NUMBER OF WORDS TO BE
                966  :                          DUPLICATED IS A USER SPECIIFED
                967  :                          OPTION DEPENDENT UPON THE
                968  :                          MESSAGE  STRUCTURE . AS A
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

               969   :                          MINIMUM THE TO/FROM ADDRESSES
               970   :                          MUST BE DUPLICATED.---
0420 ED80      971           LDIR               ;DUPLICATE N WORDS
0422 323A38    972           LD   (RXUR01+2),A  ;MESSAGE LENGTH EQUAL DUPLICATED NUMBER
               973   :                          OF WORDS+1(THE INPUTTED CHAR-
               974   :                          ACTER STILL TO BE STORED)
0425 F1        975           POP  AF            ;A=INPUTTED WORD
0426 12        976           LD   (DE),A        ;STORE THE WORD
0427 213C38    977           LD   HL,RXJR01+4   ;HL=ADDRESS OF CONTROL WORD
042A CBE6      978           SET  4,(HL)        ;SET MESSAGE SEQUENCE BIT
042C 3E01      979           LD   A,1           ;A=1
042E 86        980           ADD  A,(HL)        ;INCREMENT CONTROL WORD SEQUENCE NUMBER
042F C39803    981           JP   SPOG01        ;JUMP TO RETURN SECTION
               982   ;*****************************************************************
               983   ;**********    SUBROUTINE TRANSMIT QUEUE     ***  ***********
               984   ;*****************************************************************
               985   ;THIS SUBROUTINE PUT THE MESSAGE STORED IN THE MEMORY BLOCK
               986   ;ONTO THE NETWORK TO-BE-TRANSMITTED QUEUE. A CIRCULAR QUEUE
               987   ;(NWTXQ) IS USED TO STORE THE ADDRESS OF THE MESSAGE'S
               988   ;MEMORY BLOCK. THE ADDRESS OF THE MESSAGE MEMORY BLOCK MUST
               989   ;BE IN THE BC REGISTER. A CHECK IS ALSO MADE TO DETERMINE
               990   ;IF PROCESSOR #2 IS USING THE QUEUE. IF SO, A WAIT LOOP IS
               991   ;ENTERED.
               992   ;                          LABELS
               993   ;NWTXQ-  NAME OF SUBROUTINE
               994   ;NTXQFR- PROVIDES USAGE STATUS OF NETWORK TRANSMIT QUEUE
               995   ;NWTXEQ- ADDRESS OF NETWORK TRANSMIT END OF QUEUE
               996   ;NWTXHQ- ADDRESS OF NETWORK TRANSMIT HEAD OF QUEUE
               997   ;NWTXSQ- ADDRESS OF NETWORK TRANSMIT START OF QUEUE
               998   ;NWTXTQ- ADDRESS OF NETWORK TRANSMIT TAIL OF QUEUE
               999   ;                        REGISTERS
              1000   ;ADDRESS OF THE MESSAGE MEMORY BLOCK MUST BE IN THE BC
              1001   ;REGISTER PRIOR TO CALLING THIS SUBROUTINE
0432 215321   1002   NWTXQ  LD   HL,NTXQFR     ;ADDRESS OF NETWORK QUEUE STATUS WORD
0435 CBCE     1003           SET  1,(HL)        ;STATUS WORD TO PROCESSOR #1 WAITING
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

0437 CB56      1004           BIT  2,(HL)        ;CHECK IF PROCESSOR #2 WAITING
0439 CBC6      1005           SET  0,(HL)        ;SET STATUS WORD TO PROCESSOR #1 USING
043B 2A5621    1006           LD   HL,(NWTXTQ)   ;HL=ADDRESS OF TAIL OF QUEUE
043E           1007           ADDTQ NWTXSQ,NWTXEQ,NWTXTQ
043E 2A5621    1007 +         LD   HL,(NWTXTQ)   ;HL=ADDRESS OF TAIL OF QUEUE
0441 71        1007 +         LD   (HL),C        ;PUT LOW ORDER BYTE OF THE MESSAGE
               1007 +;                            MEMORY BLOCK AT TAIL OF QUEUE
0442 23        1007 +         INC  HL            ;PUT HIGH ORDER BYTE OF THE MESSAGE
0443 70        1007 +         LD   (HL),B        ;MEMORY BLOCK AT TAIL OF QUEUE
0444 23        1007 +         INC  HL            ;HL=NEW TAIL OF QUEUE
0445 E5        1007 +         PUSH HL            ;SAVE NEW TAIL OF QUEUE ADDRESS
0446 115822    1007 +         LD   DE,NWTXEQ     ;DE=ADDRESS OF END OF QUEUE
0449 ED52      1007 +         SBC  HL,DE         ;HL=CURRENT LOCATION-END QUEUE-CARRY
044B FA5104    1007 +         JP   M,A_0002      ;IF SUBTRACTION NEGATIVE JUMP
044E 215821    1007 +         LD   HL,NWTXSQ     ;HL=ADDRESS OF START OF QUEUE
0451 225621    1007 +A_0002   LD   (NWTXTQ),HL   ;STORE VALUE OF HL INTO TAIL ADDRESS
0454 215321    1008           LD   HL,NTXQFR     ;LOCATION OF QUEUE STATUS WORD
0457 CB86      1009           RES  0,(HL)        ;SET STATUS WORD TO PROCESSOR #1
0459 CB8E      1010           RES  1,(HL)        ;NOT WAITING OR USING
045B C9        1011           RET
               1012  ;*****************************************************************
               1013  ;*************      SUBROUTINE LOCAL QUEUE        ****************
               1014  ;*****************************************************************
               1015  ;THIS SUBROUTINE PUT THE MESSAGE STORED IN THE MEMORY BLOCK
               1016  ;ONTO THE LOCAL TRANSMIT QUEUE. A CIRCULAR QUEUE
               1017  ;(LOTXQ) IS USED TO STORE THE ADDRESS OF THE MESSAGE'S
               1018  ;MEMORY BLOCK. THE ADDRESS OF THE MESSAGE MEMORY BLOCK MUST
               1019  ;BE IN THE BC REGISTER. A CHECK IS ALSO MADE TO DETERMINE
               1020  ;IF PROCESSOR #2 IS USING THE QUEUE. IF SO, A WAIT LOOP IS
               1021  ;ENTERED
               1022  ;                          LABELS
               1023  ;LOTXQ -NAME OF SUBROUTINE
               1024  ;LTXQFR-PROVIDE USAGE STATUS OF LOCAL TRANSMIT QUEUE
               1025  ;LOTXEQ-ADDRESS OF LOCAL TRANSMIT END OF QUEUE
               1026  ;LOTXHQ-ADDRESS OF LOCAL TRANSMIT HEAD OF QUEUE
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

               1027  :LOTXSQ-ADDRESS OF LOCAL TRANSMIT START OF QUEUE
               1028  :LOTXTQ-ADDRESS OF LOCAL TRANSMIT TAIL OF QUEUE
               1029  :                          REGISTERS
               1030  :ADDRESS OF THE MESSAGE MEMORY BLOCK MUST BE IN THE BC
               1031  :REGISTER PRIOR TO CALLING THIS SUBROUTINE
045C 214A20    1032  LOTXQ    LD   HL,LTXQFR     :ADDRESS OF LOCAL QUEUE STATUS WORD
045F CBCE      1033           SET  1,(HL)        :STATUS WORD TO PROCESSOR #1 WAITING
0461 CB56      1034           BIT  2,(HL)        :CHECK IF PROCESSOR #2 WAITING
0463 C25C04    1035           JP   NZ,LOTXQ      :JUMP IF PROCESSOR #2 USING
0466 CBC6      1036           SET  0,(HL)        :SET STATUS WORD TO PROCESSOR #1 USING
0468           1037           ADDTQ LOTXSQ,LOTXEQ,LOTXTQ
0468 2A4D20    1037 +         LD   HL,(LOTXTQ)   :HL=ADDRESS OF TAIL OF QUEUE
046B 71        1037 +         LD   (HL),C        :PUT LOW ORDER BYTE OF THE MESSAGE
               1037 +:                            MEMORY BLOCK AT TAIL OF QUEUE
046C 23        1037 +         INC  HL            :PUT HIGH ORDER BYTE OF THE MESSAGE
046D 70        1037 +         LD   (HL),B        :MEMORY BLOCK AT TAIL OF QUEUE
046E 23        1037 +         INC  HL            :HL=NEW TAIL OF QUEUE
046F E5        1037 +         PUSH HL            :SAVE NEW TAIL OF QUEUE ADDRESS
0470 115221    1037 +         LD   DE,LOTXEQ     :DE=ADDRESS OF END OF QUEUE
0473 ED52      1037 +         SBC  HL,DE         :HL=CURRENT LOCATION-END QUEUE-CARRY
0475 FA7B04    1037 +         JP   M,A_0003      :IF SUBTRACTION NEGATIVE JUMP
0478 214F20    1037 +         LD   HL,LOTXSQ     :HL=ADDRESS OF START OF QUEUE
047B 224D20    1037 +A_0003   LD   (LOTXTQ),HL   :STORE VALUE OF HL INTO TAIL ADDRESS
047E 214A20    1038           LD   HL,LTXQFR     :ADDRESS OF LOCAL QUEUE STATUS WORD
0481 CB86      1039           RES  0,(HL)        :SET STATUS WORD TO PROCESSOR #1
0483 CB8E      1040           RES  1,(HL)        :NOT WAITING OR USING
0485 C9        1041           RET                :RETURN
               1042  :*****************************************************************
               1043  :**************      SUBROUTINE ALLOCATE MEMORY    ***************
               1044  :*****************************************************************
               1045  :THIS SUBROUTINE ALLOCATES A BLOCK OF MEMORY FOR MESSAGE
               1046  :STORAGE. A MEMORY TABLE (LOMNTB) IS USED TO DETERMINE
               1047  :WHERE STORAGE IS AVAILABLE. THIS MEMORY TABLE CONTAINS
               1048  :THE START ADDRESS OF ALL UNALLOCATED MEMORY BLOCKS. A
               1049  :CHECK IS MADE TO DETERMINE IF A MEMORY BLOCK IS AVAIL-
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND      COMMENT

               1050  ;ABLE. IF NOT, THE SUBROUTINE ENTERS A WAIT LOOP (ALGONE).
               1051  ;A CHECK IS ALSO MADE TO DETERMINE IF THE OTHER PROCESSOR
               1052  ;IS USING THE MEMORY TABLE. IF SO, A WAIT LOOP IS ENTERED.
               1053  ;THE ALLOCATED BLOCK ADDRESS IS RETURNED IN THE BC REGISTERS
               1054  ;
               1055  ;                          LABELS
               1056  ;ALLMEN- NAME OF SUBROUTINE
               1057  ;ALGONE- START OF WAIT LOOP IF MEMORY UNAVAILABLE
               1058  ;MNTBED- ADDRESS OF LAST ENTRY IN MEMORY TABLE
               1059  ;MNTBFR- PROVIDES USAGE STATUS OF MEMORY TABLE
               1060  ;MNTBPT- MEMORY TABLE ADDRESS OF NEXT FREE MEMORY BLOCK
               1061  ;
               1062  ;                         REGISTERS
               1063  ;THE SUBROUTINE DOES NOT SAVE ANY REGISTERS FROM THE CALL-
               1064  ;ING PROGRAM. THE SUBROUTINE USES REGISTERS BC,HL,DE. THE
               1065  ;BLOCK ADDRESS IS RETURNED IN REGISTER BC
               1066  ;
               1067  ;                       SPECIAL INST
               1068  ;SINCE THIS SUBROUTINE CAN PUT THE OTHER PROCESSOR INTO A
               1069  ;WAIT LOOP, THE CALLING ROUTINE SHOULD DISABLE INTERRUTS
               1070  ;BEFORE CALLING THE SUBROUTINE. IF THE ALGONE PORTION OF
               1071  ;THE SUBROUTINE IS EXECUTED THE ROUTINE DISABLES INTERRUPTS
0486 21401F    1072  ALLMEN  LD    HL,MNTBFR     ;LOCATION OF MEMORY TABLE STATUS WORD
0489 CBCE      1073          SET   1,(HL)        ;STATUS WORD TO PROCESSOR #1 WAITING
048B CB56      1074          BIT   2,(HL)        ;CHECK IF PROCESSOR #2 WAITING
048D C28604    1075          JP    NZ,ALLMEN     ;REPEAT LOOP IF PROCESSOR #2 WAITING
0490 CBC6      1076          SET   0,(HL)        ;SET STATUS WORD TO PROCESSOR #1 USING
0492 2A411F    1077          LD    HL,(MNTBPT)   ;TABLE ADD OF NEXT FREE MEMORY BLOCK
0495 5D        1078          LD    E,L           ;SAVE THE CONTENTS
0496 54        1079          LD    D,H           ;OF HL IN DE
0497 014620    1080          LD    BC,MNTBED     ;ADDRESS OF LAST ENTRY IN MEMORY TABLE
049A ED42      1081          SBC   HL,BC         ;HL=CURRENT LOCATION-LAST ENTRY-CARRY
049C F2AF04    1082          JP    P,ALGONE      ;POSTIVE ALL MEMORY ALLOCATED SO JUMP
049F EB        1083          EX    DE,HL         ;TABLE ADD OF NEXT FREE MEMORY BLOCK
04A0 4E        1084          LD    C,(HL)        ;C=LOWER BYTE OF FREE MEMORY BLOCK ADD
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

04A1 23        1085          INC  HL
04A2 46        1086          LD   B,(HL)        ;B=HIGH BYTE OF FREE MEMORY BLOCK ADD
04A3 23        1087          INC  HL
04A4 22411F    1088          LD   (MNTBPT),HL  ;NEW TABLE ADDRESS STORED POINTER WORD
04A7 21401F    1089          LD   HL,MNTBFR    ;LOCATION OF MEMORY TABLE STATUS WORD
04AA CB86      1090          RES  0,(HL)       ;SET STATUS WORD TO PROCESSOR #1
04AC CB8E      1091          RES  1,(HL)       ;NOT WAITING OR USING
04AE C9        1092          RET               ;RETURN TO CALLING PROGRAM
04AF E1        1093  ALGONE  POP  HL           ;HL=ADDRESS OF MEMORY TABLE STATUS WORD
04B0 CB86      1094          RES  0,(HL)       ;SET STATUS WORD TO PROCESSOR #1
04B2 CB8E      1095          RES  1,(HL)       ;NOT WAITING OR USING
04B4 F5        1096          PUSH AF           ;SAVE REG A CALLING PROGRAM
04B5 FB        1097          EI                ;ENABLE ANY DISABLED INTERRUPTS
04B6 2A411F    1098  ALGON1  LD   HL,(MNTBPT)  ;TABLE ADD OF NEXT FREE MEMORY BLOCK
04B9 ED42      1099          SBC  HL,BC        ;HL=CURRENT LOCATION-LAST ENTRY-CARRY
04BB F2B604    1100          JP   P,ALGON1     ;IF NEGATIVE REPEAT LOOP
04BE F3        1101          DI                ;REENABLE THE DISABLED INTERRUPTS
04BF F1        1102          POP  AF           ; RESTORE REG A CALLING PROGRAM
04C0 C38604    1103          JP   ALLMEN       ;JUMP TO START OF SUBROUTINE
               1104  ;*****************************************************
               1105  ;**********   SUBROUTINE DEALLOCATE MEMORY   *** **********
               1106  ;*****************************************************
               1107  ;THIS SUBROUTINE DEALLOCATES A BLOCK OF MEMORY. A MEMORY
               1108  ;TABLE (LOMNTB) IS USED TO RETURN THE MEMORY BLOCK ADDRESS
               1109  ;TO THIS TABLE. THIS MEMORY TABLE CONTAINS THE START ADDRESS
               1110  ;OF ALL UNALLOCTED MEMORY BLOCKS. THE ADDRESS OF THE MEMORY
               1111  ;BLOCK TO BE PUT INTO THE MEMORY TABLE MUST BE IN THE BC
               1112  ;REGISTERS. THE SUBROUTINE CHECKS TO DETERMINE IF THE MEMORY
               1113  ;TABLE IS BEING USED. IF SO A WAIT LOOP IS ENTERED
               1114  ;                          LABELS
               1115  ;DALMEN- NAME OF SUBROUTINE
               1116  ;MNTBFR- PROVIDES USAGE STATUS OF MEMORY TABLE
               1117  ;MNTBPT- MEMORY TABLE ADDRESS OF NEXT FREE MEMORY BLOCK
               1118  ;                        REGISTERS
               1119  ;THE SUBROUTINE DOES NOT SAVE ANY REGISTERS FROM THE CALL-
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

               1120  :PROGRAM. THE SUBROUTINE USES REGISTERS BC,H.,DE.
               1121  :                        SPECIAL INST
               1122  :SINCE THIS SUBROUTINE CAN PUT PROCESSOR #2 INTO A WAIT
               1123  :LOOP, THE CALLING ROUTINE SHOULD DISABLE INTERRUPTS
               1124  :BEFORE CALLING THE SUBROUTINE
04C3 21401F    1125  DALMEN  LD    HL,MNTBER     :LOCATION OF MEMORY TABLE STATUS WORD
04C6 CBCE      1126          SET   1,(HL)        :STATUS WORD TO PROCESSOR #1 WAITING
04C8 CB56      1127          BIT   2,(HL)        :CHECK IF PROCESSOR #2 WAITING
04CA C2C304    1128          JP    NZ,DALMEN     :REPEAT LOOP IF PROCESSOR #2 WAITING
04CD CBC6      1129          SET   0,(HL)        :SET STATUS WORD TO PROCESSOR #1 USING
04CF 2B        1130          DEC   HL            :HIGH ORDER BYTE OF THE MEMORY BLOCK
04D0 70        1131          LD    (HL),B        :ADDRESS INTO THE MEMORY TABLE
04D1 2B        1132          DEC   HL            :LOW ORDER BYTE OF THE MEMORY BLOCK
04D2 71        1133          LD    (HL),C        :ADDRESS INTO THE MEMORY TABLE
04D3 22411F    1134          LD    (MNTBPT),HL   :NEW TABLE ADD STORED IN POINTER WORD
04D6 69        1135          LD    L,C           :L=LOW ORDER BYTE OF MEMORY BLOCK ADD
04D7 60        1136          LD    H,B           :H=HIGH ORDER BYTE OF MEMORY BLOCK ADD
04D8 21401F    1137          LD    HL,MNTBER     :LOCATION OF MEMORY TABLE STATUS WORD
04DB CB86      1138          RES   0,(HL)        :SET STATUS WORD TO PROCESSOR #1
04DD CB8E      1139          RES   1,(HL)        :NOT WAITING OR USING
04DF C9        1140          RET                 :RETURN TO CALLING PROGRAM
               1141          ORG   4032D
               1142  :**** THIS SECTION DEFINES STORAGE SPACE FOR THE INTERRUPT ******
               1143  :VECTOR TABLE.
0FC0 8C03      1144  INVTAB  DEFW SARX01
0FC2 2E03      1145          DEFW SATX01
0FC4           1146          DEFS 252D
               1147  :**** THIS SECTION DEFINES THE LOCATION OF THE STACK POINTER*****
10C0 301F      1148  SPLOC   DEFW 8000-15D
               1149  :**** THIS SECTION USES THE ASSEMBLER TO ALLOCATE AND     *****
               1150  :DEFINE THE STORAGE SPACE REQUIRED FOR THE LOCAL ADDRESS
               1151  :TABLE. THE ENTRIES IN THE TABLE CONSIST OF THE I/O PORT
               1152  :ADDRESS PLUS THE LOCATION USED TO STORE THE ADDRESS OF A
               1153  :MEMORY BLOCK TO BE TRANSMITTED LOCALLY.
10C2 00        1154  LOADTB  DEFB 0
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND      COMMENT

10C3 3038      1155          DEFW TXUR01
               1156  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE AND     ******
               1157  :DEFINE THE STORAGE SPACE REQUIRED TO SUPPORT THE INITIAL-
               1158  :IZATION LINK LIST FOR THE DIFFERENT CHIPS.
10C5           1159  PIC1    DEFS 7
10CC           1160  UART00  DEFS 9
10D5 D710      1161          DEFW UART01
10D7           1162  UART01  DEFS 11
10E2           1163  SART    DEFS 13
10EF           1164  PIO1    DEFS 3
10F2           1165  SART00  DEFS 14
               1166  :**** THIS SECTION DEFINES THE VARIABLES ASSOCIATED WITH   *****
               1167  :THE MEMORY INITIALIZATION SECTION.
1100 80        1168  BLKSIZ  DEFB 128D
1101 80        1169  BLKNUM  DEFB 128D
1102           1170  MENST   DEFS 10000D
               1171  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE THE     *****
               1172  :THE STORAGE SPACE FOR THE MULTIBUFFER ASSEMBLY AREA. TWO
               1173  :MULTIBUFFER STORAGE AREAS ARE CREATED BY THE ASSEMBLIER.
3812           1174  MBSA01  DEFS 19D
3825           1175  MBSA02  DEFS 19D
               1176  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE THE     *****
               1177  :STORAGE SPACE REQUIRED TO SUPPORT THE USART TRANSMIT
               1178  :ROUTINE AND RECEIVE ROUTINE
3838           1179  RXUR01  DEFS 5D
383D           1180  TXUR01  DEFS 8D
               1181  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE THE     *****
               1182  :STORAGE SPACE REQUIRED FOR THE LOCAL BUSY TRANSMIT QUEUE.
               1183  :THE END OF QUEUE ADDRESS MUST BE THE ACTUAL END OF QUEUE
               1184  :ADDRESS. TO DO THIS THE TOTAL QUEUE STORAGE SPACE IS
               1185  :ALLOCATED BY THREE ASSEMBLIER INSTRUCTIONS. STORAGE SPACES
               1186  :ARE ALSO ALLOCATED FOR THE POINTERS TO THE HEAD OF THE
               1187  :QUEUE AND TAIL OF THE QUEUE.
3845           1188  LBTXHQ  DEFS 2
3847           1189  LBTXTQ  DEFS 2
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND      COMMENT

3849           1190  LBTXSQ   DEFS 2*128+2
3948           1191           DEFS 1
394C           1192  LBTXEQ   DEFS 1
               1193           ORG  8000D
               1194  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE THE     *****
               1195  :STORAGE SPACE REQUIRED FOR THE MEMORY TABLE.
               1196  :THE END OF TABLE ADDRESS MUST BE THE ACTUAL END OF TABLE
               1197  :ADDRESS. TO DO THIS THE TOTAL QUEUE STORAGE SPACE IS
               1198  :ALLOCATED BY THREE ASSEMBLIER INSTRUCTIONS. STORAGE SPACES
               1199  :ARE ALSO ALLOCATED FOR THE POINTER TO THE HEAD OF THE
               1200  :TABLE AND THE TABLE STATUS WORD.
1F40           1201  MNTBFR   DEFS 1
1F41           1202  MNTBPT   DEFS 2
1F43           1203  LOMNTB   DEFS 2*128+2
2045           1204           DEFS 1
2046           1205  MNTBED   DEFS 1
               1206  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE AND     *****
               1207  :DEFINE THE STORAGE SPACE REQUIRED FOR THE NETWORK ADDRESS
               1208  :TABLE THIS TABLE CONTAINS THE NETWORK ADDRESS OF ALL
               1209  :PERIPHERALS CONNECTED TO THE UNIVERSAL NETWORK INTERFACE
               1210  :DEVICE. IN ADDITION THE NUMBER OF ADDRESSES IN THE TABLE
               1211  :IS STORED AT LOCATION NWTBNU.
2047 41        1212  NWADTB   DEFB 'A'
2048 31        1213           DEFB '1'
2049 02        1214  NWTBNU   DEFB 2
               1215  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE THE     *****
               1216  :STORAGE SPACE REQUIRED FOR THE LOCAL TRANSMIT QUEUE.
               1217  :THE END OF QUEUE ADDRESS MUST BE THE ACTUAL END OF QUEUE
               1218  :ADDRESS. TO DO THIS THE TOTAL QUEUE STORAGE SPACE IS
               1219  :ALLOCATED BY THREE ASSEMBLIER INSTRUCTIONS. STORAGE SPACES
               1220  :ARE ALSO ALLOCATED FOR THE POINTERS TO THE HEAD OF THE
               1221  :QUEUE AND TAIL OF THE QUEUE AND THE QUEUE STATUS WORD.
204A           1222  LTXQFR   DEFS 1
204B           1223  LOTXHQ   DEFS 2
204D           1224  LOTXTQ   DEFS 2
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND      COMMENT

204F           1225  LOTXSQ  DEFS 2*128+2
2151           1226          DEFS 1
2152           1227  LOTXEQ  DEFS 1
               1228  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE THE     *****
               1229  :STORAGE SPACE REQUIRED FOR THE NETWORK TRANSMIT QUEUE.
               1230  :THE END OF QUEUE ADDRESS MUST BE THE ACTUAL END OF QUEUE
               1231  :ADDRESS. TO DO THIS THE TOTAL QUEUE STORAGE SPACE IS
               1232  :ALLOCATED BY THREE ASSEMBLIER INSTRUCTIONS. STORAGE SPACES
               1233  :ARE ALSO ALLOCATED FOR THE POINTERS TO THE HEAD OF THE
               1234  :QUEUE AND TAIL OF THE QUEUE AND THE QUEUE STATUS WORD.
2153           1235  NTXQFR  DEFS 1
2154           1236  NWTXHQ  DEFS 2
2156           1237  NWTXTQ  DEFS 2
2158           1238  NWTXSQ  DEFS 2*128D+2
225A           1239          DEFS 1
225B           1240  NWTXEQ  DEFS 1
225C           1241          END

TOTAL ASSEMBLER ERRORS =     0
```

SYMBOL TABLE

ALGON1	0486	ALGONE	04AF	ALLMEN	0486	A_0002	0451
A_0003	047B	BLKNUM	1101	BLKSIZ	1100	B_0001	0163
DALMEN	04C3	DELE01	03D4	DHLOG	014E	ENDPIC	0035
ENDPIO	00C3	ENDSYN	009A	EO401	03DB	EXMN01	03E3
GTMN01	0386	INVTAB	0FC0	ISTART	0041	ITMEN	0005
ITJART	0316	LBTXE0	396C	LBTXH0	3B4E	LBTXS0	38+3
LBTXT0	3847	LOADTB	10C2	LO1NTB	1F43	LOTXE0	2152
LOTXH0	204B	LOTX0	045C	LOTXS0	204E	LOTXT0	2040
LTXOER	204A	MAIN	00E7	MAIN01	00E8	MAIN02	0141
MAIN03	017A	MAIN04	01C3	MAIN0B	01E6	MAIN3A	01A0
MAIN3B	01A2	MAIN3C	01B8	MAIN4A	01D2	MAIN4B	01D3
MBSA01	3812	MBSA02	382E	MBTXCK	0108	MBTX_0	0113
MENST	1102	MNTRE0	204C	MNTRER	1F40	MNTRST	1F41
MULT01	0214	MULT02	0279	MULT03	02C7	MULT1A	0233
MULT2A	028E	MULT2B	0285	MULT2C	0283	MULT2D	0230
NDMN01	03C1	NONSY1	008C	NONSYN	006C	NTXOER	2153
NWADTB	2047	NWTRNU	2049	NWTXE0	2259	NWTXH0	2154
NWTX0	0432	NWTXS0	215E	NWTXT0	215C	PIC1	1005
PICIA	00AF	PICIB	009D	PIO1	10EF	RXUR01	3839
SART	10E2	SART00	10F2	SARX01	038C	SATX01	032E
SPLOC	10C0	SPOG01	0388	SYNC	0074	SYNC1	0035
SYNC2	0077	TXBUS1	02F6	TXBUS2	02F7	TXBUS3	0303
TXBUSY	02DD	TXED01	0352	TXED1A	0369	TXED1B	0358
TXED1C	0383	TXRT01	034A	TXUR01	383D	UART00	1003
UART01	1007						

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND      COMMENT

                 1  ;////////////////////////////////////////////////////////////
                 2  ;///////////////       OPERATING SYSTEM #2       ////////////////
                 3  ;////////////////////////////////////////////////////////////
                 4  ;THIS ROUTINE ESTABLISHES THE OPERATING SYSTEM FOR PROCESS-
                 5  ;OR #2 OF THE UNIVERSAL NETWORK INTERFACE. THIS PROCESSOR
                 6  ;IS ASSIGNED THE FUNCTIONS ASSOCIATED WITH INTERFACING THE
                 7  ;NETWORK TO THE NETWORK INTERFACE. THE OPERATING SYSTEM
                 8  ;IS COMPOSED OF TWO MAJOR AREAS. THE FIRST AREA CONSISTS OF
                 9  ;A NUMBER OF SECTIONS WHICH INITIALIZE THE DIFFERENT CHIPS
                10  ;ASSOCIATED WITH PROCESSOR BOARD #2 AND THE NETWORK
                11  ;CARDS. THE SECOND SECTION ACTUALLY ACCOMPLISHES THE INTER-
                12  ;FACING FUNCTION. THE DIFFERENT AREAS AND SECTIONS ARE
                13  ;EXPLAINED AS THEY ARE ENCOUNTERED WITHIN THE OPERATING
                14  ;SYSTEM
                15  ;
                16  ;THIS SECTION INITIALIZES THE PROCESSOR BOARD CENTRAL  ******
                17  ;PROCESSING UNIT. IT ESTABLISHES THE HIGH ORDER BYTE OF
                18  ;THE VECTOR INTERRUPT TABLE (ITVTAB), THE LOCATION IN
                19  ;MEMORY OF THE STACK POINTER (SPLOCN), THE INTERRUPT
                20  ;MODE FOR THE PROCESSOR AND THE VALUE FOR THE REFRESH REG
0000 3E40       21          LD   A,64          ;A=64D
0002 ED4F       22          LD   R,A           ;SET REFRESH REG TO 64D
0004 C34100     23          JP   NSTART        ;JUMP OVER RESTART AREA
                24          ORG  65D
0041 21C00F     25  NSTART  LD   HL,INVTAB     ;HL=ADDRESS OF VECTOR ADDRESS TABLE
0044 7C         26          LD   A,H           ;A=HIGH BYTE VECTOR ADDRESS TABLE
0045 ED47       27          LD   I,A           ;I=HIGH BYTE VECTOR ADDRESS TABLE
0047 DD2A0810   28          LD   IX,(SPLOCN)   ;IX=MEMORY ADDRESS OF STACK POINTER
004B DDF9       29          LD   SP,IX         ;LOAD THE STACK POINTER WITH IX
004D ED56       30          IM   1             ;SET INTERRUPT MODE TO VECTOR ADDR MODE
                31  ;****THIS SECTION INITIALIZES THE NON-SYNCHRONOUS ON BOARD ******
                32  ;USART. THE USART IS SET UP FOR 300 BAUD ASYNCHRONOUS OPERATION.
                33  ;THE USART UTILIZES THE PORT A CHANNEL OF THE PROCESSOR BOARD
                34  ;PARALLEL INPUT/OUTPUT CHIP FOR INTERRUPT MONITORING. THIS PORT
                35  ;MONITORS THE USART AND GENERATES AN INTERRUPT TO THE CPU IN
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND      COMMENT

                 36  :RESPONSE TO A USART INTERRUPT. THE PORT THEN PROVIDES THE LOWER
                 37  :ORDER BYTE VECTOR TABLE ADDRESS OF THE USART SERVICE ROUTINE.
                 38  :THIS SECTION ALSO INITIALIZES THE PORT A PIO CHANNEL TO
                 39  :ACCOMPLISH THIS FUNCTION. THE Z80A-CTC CHANNEL WHICH PROVIDES
                 40  :THE FREQUENCY FOR THE USART IS ALSO INITIALIZES BY THIS SECTION.
                 41  :THE I/O ADDRESSES WITHIN THIS SECTION CORRESPOND TO THE I/O
                 42  :ADDRESSES OF THE PROCESSOR BOARD AS RECEIVED FROM THE MAN-
                 43  :UFACTURE.
004F 3AFC0F      44          LD   A,(INVTAN+60) :LOAD LOW BYTE OF INTERRUPT VECTOR
0052 D3DA        45          OUT  (218D),A      :VECTOR TABLE ADDRESS TO PORT A
0054 3ECF        46          LD   A,11001111B   :SELECT MODE 3 (BIT MONITORING)
0056 D3DA        47          OUT  (218D),A      ;FOR PORT A OF PIO
0058 3E80        48          LD   A,10000000B   :SET BIT 7 OF PORT A
005A D3DA        49          OUT  (218D),A      ;AS AN INPUT PORT
005C 3E87        50          LD   A,10000111B   :SET BIT 7 TO MONITOR
005E D3DA        51          OUT  (218D),A      :FOR A LOW LEVEL
0060 3ECA        52          LD   A,11001010B   :SET USART FOR 7 BIT CHAR LENGTH, NO
0062 D3DF        53          OUT  (223D),A      ;PARITY, TWO STOP BITS, 16X BAUD FACTOR
0064 3E26        54          LD   A,00100110B   :DISABLE USART TX, DTR=1, ENABLE USART
0066 D3DF        55          OUT  (223D),A      :RX, RTS=1 , NO ERROR OR INTERNAL RESET
0068 3E15        56          LD   A,00010101B   :SET CTC CHANNEL 1 TO TIMER MODE,
006A D3D5        57          OUT  (213D),A      :DIVIDE BY 16 IN PRESCALER
006C 3E10        58          LD   A,00010000B   :SET CTC CHANNEL 1 TIME CONSTANT = 16
006E D3D5        59          OUT  (213D),A      :WHICH RESULTS IN 300 BAUD OPERATION
                 60  :****THIS SECTION INITIALIZES THE Z80A-SIO CHIP ASSOCIATED ******
                 61  :WITH THE NETWORK CARD. THE INITIALIZATION IS ACCOMPLISHED
                 62  :THROUGH THE USE OF A PARAMETER LIST. THIS LIST CONTAINS THE
                 63  :DIFFERENT REGISTER VALUES NEEDED TO PROGRAM THE SIO TO THE
                 64  :CONFIGURATION DESIRED. THIS SECTION OUTPUTS THE REGISTER
                 65  :ADDRESS AND THEN THE REGISTER. THE LIST IS AS FOLLOWS;
                 66  :SIOXX    -I/O ADDRESS OF PORT A COMMAND
                 67  :      +1 -WORDS TO BE TRANSMITTED
                 68  :      +7 -TO THE PORT A COMMAND REGISTERS
                 69  :      +8 -WORDS TO BE TRANSMITTED
                 70  :      +15-TO THE PORT B COMMAND REGISTERS
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

                 71  ;      +16-ADDRESS OF NEXT
                 72  ;      +17-PARAMETER LIST
0070 21D710      73           LD   HL,SIO01      ;ADDRESS OF SIO PARAMETER LIST
0073 4E          74  REPT1A   LD   C,(HL)        ;C=I/O ADDRESS PORT A COMMAND/STATUS
0074 23          75           INC  HL            ;HL=FIRST WORD IN PARAMETER LIST
0075 1603        76           LD   D,003D        ;D=REGISTER ADDRESS 3
0077 0606        77           LD   B,006D        ;B=NUMBER OF REGISTERS
0079 EDA3        78           OUTI               ;OUTPUT REGISTER 0 WORD
007B 3E01        79           LD   A,001D        ;A=REGISTER ADDRESS 1
007D ED79        80           OUT  (C),A         ;OUTPUT REGISTER ADDRESS
007F EDA3        81           OUTI               ;OUTPUT REGISTER 1 WORD
0081 ED51        82  REPT01   OUT  (C),D         ;OUTPUT NEXT REGISTER ADDRESS
0083 14          83           INC  D             ;D=D+1
0084 EDA3        84           OUTI               ;OUTPUT NEXT REGISTER WORD
0086 C28100      85           JP   NZ,REPT01     ;REPEAT UNTIL B=0
0089 1601        86           LD   D,001D        ;D=REGISTER ADDRESS 1
008B 0607        87           LD   B,007D        ;B=NUMBER OF REGISTERS
008D 0C          88           INC  C             ;C=I/O ADDRESS PORT B COMMAND/STATUS
008E EDA3        89           OUTI               ;OUTPUT REGISTER 0 WORD
0090 ED51        90  REPT02   OUT  (C),D         ;OUTPUT NEXT REGISTER ADDRESS
0092 14          91           INC  D             ;D=NEXT REGISTER ADDRESS
0093 EDA3        92           OUTI               ;OUTPUT NEXT REGISTER WORD
0095 C29000      93           JP   NZ,REPT02     ;REPEAT UNTIL B=0
0098 4E          94           LD   C,(HL)        ;C=LOW BYTE PARAMETER LIST ADDRESS
0099 23          95           INC  HL            ;B=HIGH BYTE NEXT
009A 46          96           LD   B,(HL)        ;PARAMETER LIST ADDRESS
009B AF          97           XOR  A             ;A=ZERO
009C B0          98           ADD  A,B           ;A=HIGH BYTE PARAMETER LIST ADDRESS
009D CAA500      99           JP   Z,ENDSIO      ;IF ZERO END OF PARAMETER LIST SO JUMP
00A0 69         100           LD   L,C           ;SET HL TO THE ADDRESS
00A1 60         101           LD   H,B           ;OF THE NEXT PARAMETER LIST
00A2 C37300     102           JP   REPT1A        ;REPEAT THE LOOP
                103  ;****THIS SECTION INITIALIZES THE Z80A-CTC CHANNELS ASSOC- ******
                104  ;IATED WITH THE NETWORK CARD. THE INITIALIZATION IS ACCOMPLISHED
                105  ;THROUGH THE USE OF A PARAMETER LIST. FOR EACH CHANNEL THE LIST
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND      COMMENT

                106  :CONTAINS A MODE WORD AND THEN A TIMER PRESCALER VALUE. THE
                107  :LIST IS AS FOLLOWS:
                108  :CTCXX-I/O ADDRESS OF CHANNEL ZERO
                109  :      +1 -HIGH BYTE VECTOR TABLE ADDRESS
                110  :      +2 -LOW BYTE VECTOR TABLE ADDRESS
                111  :      +3 -MODE WORD FOR CHANNEL ZERO
                112  :      +4 -PRESCALER VALUE FOR CHANNEL ZERO
                113  :      +5 -MODE WORD FOR CHANNEL ONE
                114  :      +6 -PRESCALER VALUE FOR CHANNEL ONE
                115  :      +7 -ADDRESS OF NEXT
                116  :      +8 -PARAMETER LIST
00A5 21E810     117  ENDSIO  LD   HL,CTC014    :ADDRESS OF PARAMETER TABLE
00A8 4E         118  CTC01   LD   C,(HL)       :C=CHANNEL 0 I/O ADDRESS
00A9 23         119          INC  HL           ;HL=ADDRESS OF THE VECTOR TABLE
00AA 23         120          INC  HL           :INTERRUPT ADDRESS
00AB 5E         121          LD   E,(HL)       :E=LOW BYTE VECTOR TABLE ADDRESS
00AC ED59       122          OUT  (C),E        :OUTPUT VECTOR ADDRESS TO CHANNEL 0
00AE EDA3       123          OUTI              :SET OPERATING MODE
00B0 EDA3       124          OUTI              :SET TIME CONSTANT
00B2 0C         125          INC  C            :C=CHANNEL 1 I/O ADDRESS
00B3 EDA3       126          OUTI              :SET OPERATING MODE
00B5 EDA3       127          OUTI              :SET TIME CONSTANT
00B7 4E         128          LD   C,(HL)       :C=LOW BYTE PARAMETER LIST ADDRESS
00B8 23         129          INC  HL           ;B=HIGH BYTE NEXT
00B9 46         130          LD   B,(HL)       :PARAMETER LIST ADDRESS
00BA AF         131          XOR  A            ;A=ZERO
00BB 80         132          ADD  A,B          :A=HIGH BYTE PARAMETER LIST ADDRESS
00BC CAC400     133          JP   Z,ENDCTC     :IF ZERO END OF PARAMETER LIST SO JUMP
00BF 69         134          LD   L,C          :SET HL TO THE ADDRESS
00C0 60         135          LD   H,B          ;OF THE NEXT PARAMETER LIST
00C1 C3A800     136          JP   CTC01        :REPEAT THE LOOP
                137  :*****THIS SECTION INITIALIZES THE DIFFERENT QUEUES USED BY******
                138  :THE NETWORK BOARD PROCESSOR. THESE QUEUES ARE AS FOLLOWS:
                139  :          NWTXSQ  -NETWORK TRANSMIT QUEUE START ADDRESS
                140  :          NWTXEQ  -NETWORK TRANSMIT QUEUE END ADDRESS.
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND      COMMENT

               141   :          NWRXSQ  -NETWORK RECEIVE QUEUE START ADDRESS
               142   :          NWRXEQ  -NETWORK RECEIVE QUEJE END ADDRESS
               143   :          NATXSQ  -NETWORK ALREADY TRANSMITTED QUEUE START ADD
               144   :          NATXEQ  -NETWORK ALREADY TRANSMITTED QUEUE END ADD
               145   :OTHER LABELS ASSOCIATED WITH THE QUEUE ARE:
               146   :          NWTXHQ  -ADDRESS OF HEAD OF NETWORK TRANSMIT QUEUE
               147   :          NWTXTQ  -ADDRESS OF TAIL OF NETWORK TRANSMIT QUEUE
               148   :          NWRXHQ  -ADDRESS OF HEAD OF NETWORK RECEIVE QUEUE
               149   :          NWRXTQ  -ADDRESS OF TAIL OF NETWORK RECEIVE QUEUE
               150   :          NATXHQ  -ADDRESS OF HEAD OF ALREADY TX QUEUE
               151   :          NATXTQ  -ADDRESS OF TAIL OF ALREADY TX QUEUE
               152   :EACH QUEUE IS CIRCULAR IN NATURE. THE OTHER QUEUE USED BY
               153   :THE NETWORK PROCESSOR IS THE LOCAL TRANSMIT QUEUE. THIS QUEUE
               154   :IS INITIALIZED BY THE INPUT PROCESSOR. IT'S LABELS ARE:
               155   :          LOTXSQ  -LOCAL TRANSMIT QUEUE START ADDRESS
               156   :          LOTXEQ  -LOCAL TRANSMIT QUEJE END ADDRESS
               157   :          LOTXHQ  -ADDRESS OF HEAD OF LOCAL TRANSMIT QUEUE
               158   :          LOTXTQ  -ADDRESS OF TAIL OF LOCAL TRANSMIT QUEUE
               159   :THE INITIALIZATION ROUTINE SETS THE HEAD AND TAIL OF THE QUEUES
               160   :TO THE START QUEUE ADDRESS
00C4 215821    161   ENDCTC  LD   HL,NWTXSQ    :ADDRESS OF START OF NETWORK QUEUE
00C7 225421    162           LD   (NWTXHQ),HL  :SET HEAD OF NETWORK TX QUEUE TO START
00CA 225621    163           LD   (NWTXTQ),HL  :SET TAIL OF NETWORK TX QUEUE TO START
00CD 215410    164           LD   HL,NWRXSQ    :ADDRESS OF START OF NETWORK RX QUEUE
00D0 225010    165           LD   (NWRXHQ),HL  :SET HEAD OF NETWORK RX QUEUE TO START
00D3 225210    166           LD   (NWRXTQ),HL  :SET TAIL OF NETWORK RX QUEUE TO START
00D6 210E10    167           LD   HL,NATXSQ    :ADD OF START OF NW ALREADY TX QUEUE
00D9 220A10    168           LD   (NATXHQ),HL  :HEAD OF NW ALREADY TX QUEUE TO START
00DC 220C10    169           LD   (NATXTQ),HL  :TAIL OF NW ALREADY TX QUEUE TO START
               170   :****THIS SECTION INITIALIZES THE ALTERNATIVE REGISTER SET.******
               171   :THIS REGISTER SET IS DEDICATED TO THE NETWORK RECEIVE FUNCTION.
               172   :THIS SECTION OBTAINS A BLOCK OF MEMORY FOR STORAGE OF A NET-
               173   :WORK MESSAGE. THE ADDRESS IS STORED IN THE BC REGISTERS. THE
               174   :FIRST AVAILABLE STORAGE (START ADDRESS + 1) ADDRESS IS LOADED
               175   :INTO THE HL REGISTER. THE MESSAGE LENGTH IS INITIALIZED TO
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

               176  :ZERO AND LOADED INTO THE E REGISTER.
00DF D9        177          EXX                :SWAP REGISTER SETS
00E0 CDEE02    178          CALL ALNMEN        :OBTAIN A MEMORY BLOCK STORAGE ADDRESS
00E3 60        179          LD   H,B           :HL=MEMORY BLOCK STORAGE
00E4 69        180          LD   L,C           :    START ADDRESS
00E5 23        181          INC  HL            :FIRST AVAILABLE BLOCK STORAGE ADDRESS
00E6 1E00      182          LD   E,0           :MESSAGE LENGTH = 0
00E8 D9        183          EXX                :SWAP REGISTER SETS
00E9 FB        184          EI                 :ENABLE INTERRUPT
               185  :****THIS CONCLUDES THE INITIALIZATION PART OF THE OPERATING*****
               186  :SYSTEM FOR PROCESSOR BOARD #2
               187  :
               188  :*************************************************************
               189  :**************          MACRO DEFINITION          ***********
               190  :*************************************************************
               191  :THIS SECTION DEFINES THE MACROS ADDTQ AND SUBHQ.
               192  :THESE MACROS ARE USED TO ADD A MEMORY BLOCK ADDRESS TO
               193  :THE TAIL OF A QUEUE(ADDTQ)AND TO REMOVE A MEMORY BLOCK
               194  :ADDRESS FROM THE HEAD OF A QUEUE(SUBHQ). THE VARIABLES
               195  :ASSOCIATED WITH THE MACROS ARE
               196  :          #START-ADDRESS OF THE START OF THE QUEUE
               197  :          #END  -ADDRESS OF THE END OF THE QUEUE
               198  :          #HEAD -POINTER TO THE HEAD OF THE QUEUE
               199  :          #TAIL -POINTER TO THE TAIL OF THE QUEUE
               200          LIST M
               201  ADDTQ   MACR #START,#END,#TAIL
               202          LD   HL,(#TAIL)    ;HL=ADDRESS OF TAIL OF QUEUE
               203          LD   (HL),C        ;PUT LOW ORDER BYTE OF THE MESSAGE
               204  :                           MEMORY BLOCK AT TAIL OF QUEUE
               205          INC  HL            ;PUT HIGH ORDER BYTE OF THE MESSAGE
               206          LD   (HL),B        ;MEMORY BLOCK AT TAIL OF QUEUE
               207          INC  HL            ;HL=NEW TAIL OF QUEUE
               208          PUSH HL            ;SAVE NEW TAIL OF QUEUE ADDRESS
               209          LD   DE,#END       ;DE=ADDRESS OF END OF QUEUE
               210          SBC  HL,DE         ;HL=CURRENT LOCATION-END QUEUE-CARRY
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

                211           POP  HL            :HL=ADD OF CURRENT TAIL OF QUEUE
                212           JP   M,A_#SYM      :IF SUBTRACTION NEGATIVE JUMP
                213           LD   HL,#START     :HL=ADDRESS OF START OF QUEUE
                214  A_#SYM   LD   (#TAIL),HL    :STORE VALUE OF HL INTO TAIL ADDRESS
                215           ENDM
                216  SUBHQ    MACR #START,#END,#HEAD
                217           LD   HL,(#HEAD)    :HL=ADDRESS OF HEAD OF QUEUE
                218           LD   C,(HL)        ;PUT LOW ORDER BYTE OF ADDRESS AT
                219  :                            HEAD OF QUEUE INTO REG C
                220           INC  HL            :PUT HIGH ORDER BYTE OF ADDRESS AT HEAD
                221           LD   B,(HL)        :QUEUE INTO REG B
                222           INC  HL            :HL=NEW HEAD OF QUEUE ADDRESS
                223           PUSH HL            :SAVE NEW HEAD OF QUEUE ADDRESS
                224           SBC  HL,DE         :HL=CURRENT LOCATION-END QUEUE-CARRY
                225           POP  HL            :HL=ADD OF CURRENT TAIL OF QUEUE
                226           JP   M,B_#SYM      :IF SUBTRACTION NEGATIVE JUMP
                227           LD   HL,#START     :HL=ADDRESS OF START OF QUEUE
                228  B_#SYM   LD   (#HEAD),HL    :STORE VALUE OF HL INTO TAIL ADDRESS
                229           ENDM
                230  :
                231  :****THE NEXT SECTIONS BEGINS THE SECOND PART OF THE OPER- ******
                232  :ATING SYSTEM FOR PROCESSOR #2. THE OPERATING SYSTEM MONITORS
                233  :THE NETWORK TRANSMIT QUEUE AND THE NETWORK RECEIVE
                234  :QUEUE PLUS ALLOWING ANY INTERRUPTS TO BE SERVICED. ONCE A
                235  :MESSAGE (MEMORY BLOCK ADDRESS) IS DETECTED IN ONE OF THE QUEUES
                236  :IT IS REMOVED. A MESSAGE IN THE NETWORK TX QUEUE IS TRANSMITTED
                237  :BY SUBROUTINE (NWTXSR). AFTER TRANSMISSION THE MESSAGE IS PUT
                238  :ON THE NETWORK ALREADY TRANSMITTED QUEUE AWAITING CONFORMATION
                239  :OF PROPER RECEIPT. IF THE MESSAGE WAS ON THE RECEIVE QUEUE,
                240  :THE TO ADDRESS OF THE MESSAGE IS CHECKED TO DETERMINE IF THE
                241  :MESSAGE IS FOR THE INTERFACE ITSELF OR ANY OF THE USERS
                242  :CONNECTED TO THE INTERFACE. IF NOT, THE MESSAGE IS PUT
                243  :ON THE NETWORK TRANSMIT QUEUE FOR RETRANSMISSION. IF
                244  :THE MESSAGE IS FOR A LOCAL USER, THE MESSAGE IS PLACED
                245  :ON THE LOCAL TRANSMIT QUEUE AND AN ACKNOWLEDGEMENT MESSAGE
```

```
ADDR OBJECT   STMT  LABEL   OPCD OPERAND      COMMENT

               245  :DEVELOPED AND TRANSMITTED. A MESSAGE TO THE INTERFACE
               247  :IS ASSUMED TO BE AN ACKNOWLEDGEMENT MESSAGE AN IT'S
               248  :ACKNOWLEDGEMENT IDENTIFICATION COMPARED WITH THE MESSAGE
               249  :STORED AT THE HEAD OF THE ALREADY TRANSMITTED QUEUE. THE
               250  :ALREADY TX MESSAGE IS EITHER REPLACED ONTO THE NETWORK
               251  :TRANSMIT QUEUE OR DELETED FROM MEMORY DEPENDING UPON
               252  :THE RESULTS OF THE CHECK.
               253  :                OTHER IMPORTANT LABELS
               254  :          NATXQ   -THIS IS A SUBROUTINE WHICH PUTS A BLOCK
               255  :                   STORAGE ADDRESS AT TAIL OF ALREADY TX QUEUE
               256  :          NWADTO  -ADDRESS OF START OF NETWORK ADDRESS TABLE
               257  :          NWADNU  -NUMBER OF ENTRIES IN THE NETWORK TABLE
               258  :          DNATXQ  -THIS IS A SUBROUTINE WHICH OBTAINS THE
               259  :                  -BLOCK STORAGE ADDRESS AT HEAD OF ALREADY
               260  :                  -TX QUEUE
               261  :          DNWRXQ  -THIS IS SECTION OF CODE WHICH OBTAINS THE
               262  :                   BLOCK STORAGE ADDRESS AT HEAD OF NW RX QUEUE
               263  :          DNWTXQ  -THIS IS SECTION OF CODE WHICH OBTAINS THE
               264  :                   BLOCK STORAGE ADDRESS AT HEAD OF NW TX QUEUE
               265  :          NWTXSR  -THIS IS A SUBROUTINE WHICH HANDLES THE
               266  :                   TRANSMISSION OF A NETWORK MESSAGE
               267  :                         REGISTERS
               268  :THE OPERATING SYSTEM USES THE PRIMARY REGISTER SET WITH THE
               269  :SECONDARY REGISTER SET DEDICATED TO THE NETWORK RECEIVE
               270  :FUNCTION. THE IX AND IY REGISTERS MUST BE SAVED ON THE STACK
               271  :IF USED BY OTHER SUBROUTINE
               272  :                            SPECIAL INST.
               273  :THE OPERATING SYSTEM ASSUMES ACKNOWLEDGEMENT MESSAGES ARE
               274  :NOT COMBINED WITH INFORMATION MESSAGES. OTHER CONTROL
               275  :MESSAGES SPECIFIED BY THE LINK TO LINK PROTOCOL MUST BE
               276  :INCORPORATED INTO THE OPERATING SYSTEM. THE MESSAGE
               277  :STRUCTURE ASSUMED IS
               278  :          FLAG
               279  :          DESTINATION ADDRESS
               280  :          LINK CONTROL MESSAGE ID ADRD
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

               281   :          SENDER ADDRESS
               282   :          TEXT
               283   :          ERROR CHECK
               284   :          ERROR CHECK
               285   :          FLAG
               286   :
               287   :****THIS SECTION CHECKS IF THERE IS A MESSAGE ON THE NET- ******
               288   :WORK TRANSMIT QUEUE.
00EA 2A5421    289   NMAN01   LD   HL,(NWTXHQ)   :HL=ADDRESS OF HEAD OF NETWORK TX QUEUE
00ED ED5B5621  290            LD   DE,(NWTXTQ)   :DE=ADDRESS OF TAIL OF NETWORK TX QUEUE
00F1 AF        291            XOR  A             :CARRY FLAG EQUAL ZERO
00F2 ED52      292            SBC  HL,DE         :HL=HEAD-TAIL-CARRY
00F4 C20701    293            JP   NZ,DNWTX0     :IF NOT ZERO MESSAGE ON QUEUE SO JUMP
               294   :****THIS SECTION CHECKS IF THERE IS A MESSAGE ON THE NET- ******
               295   :WORK RECEIVE QUEUE
00F7 2A5010    296   NMAN02   LD   HL,(NWRXHQ)   :HL=ADDRESS OF HEAD OF NETWORK RX QUEUE
00FA ED5B5210  297            LD   DE,(NWRXTQ)   :DE=ADDRESS OF TAIL OF NETWORK RX QUEUE
00FE AF        298            XOR  A             :CARRY FLAG EQUAL ZERO
00FF ED52      299            SBC  HL,DE         :HL=HEAD-TAIL-CARRY
0101 C27001    300            JP   NZ,DNWRX0     :IF NOT ZERO MESSAGE ON QUEUE SO JUMP
0104 C3EA00    301            JP   NMAN01        :REPEAT THE MONITORING
               302   :****THIS SECTION OF CODE OBTAINS THE MEMORY BLOCK STORAGE
               303   :ADDRESS OF MESSAGE AT HEAD OF NETWORK TRANSMIT QUEUE. A
               304   :CHECK IS FIRST MADE TO SEE IF PROCESSOR #1 IS CHANGING THE
               305   :QUEUE, IF SO A WAIT LOOP IS ENTERED. IF NOT, THE BLOCK
               306   :STORAGE ADDRESS IS OBTAINED AND PUT INTO THE BC REGISTERS.
               307   :A CHECK IS MADE TO DETERMINE IF THE HEAD ADDRESS IS AT THE
               308   :END OF THE QUEUE. IF SO IT IS REINITIALIZED TO START OF
               309   :QUEUE ADDRESS
0107 215321    310   DNWTX0   LD   HL,NTXOFR    :ADDRESS OF NETWORK QUEUE STATUS WORD
010A CBD6      311            SET  2,(HL)        :STATUS WORD TO PROCESSOR #2 WAITING
010C CB4E      312            BIT  1,(HL)        :CHECK IF PROCESSOR #1 WAITING
010E CA1601    313            JP   Z,NWOFR       :JUMP IF PROCESSOR #1 NOT WAITING
0111 CB46      314   DNWTX1   BIT  0,(HL)        :IF PROCESSOR #1 ALSO WAITING CHECK
               315   :                           IF PROCESSOR #1 USING
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

0113 C21101     316           JP   NZ,DNWTX1     ;JUMP IF PROCESSOR #1 USING
0116 CBC6       317  NWOFR    SET  0,(HL)        ;SET TO PROCESSOR #1 USING
0118 F3         318           DI                 ;DISABLE INTERRUPT
0119            319           SUBHQ NWTXSQ,NWTXEQ,NWTXHQ
0119 2A5421     319 +         LD   HL,(NWTXHQ)   ;HL=ADDRESS OF HEAD OF QUEUE
011C 4E         319 +         LD   C,(HL)        ;PUT LOW ORDER BYTE OF ADDRESS AT
                319 +:                            HEAD OF QUEUE INTO REG C
011D 23         319 +         INC  HL            ;PUT HIGH ORDER BYTE OF ADDRESS AT HEAD
011E 46         319 +         LD   B,(HL)        ;QUEUE INTO REG B
011F 23         319 +         INC  HL            ;HL=NEW HEAD OF QUEUE ADDRESS
0120 E5         319 +         PUSH HL            ;SAVE NEW HEAD OF QUEUE ADDRESS
0121 ED52       319 +         SBC  HL,DE         ;HL=CURRENT LOCATION-END QUEUE-CARRY
0123 E1         319 +         POP  HL            ;HL=ADD OF CURRENT TAIL OF QUEUE
0124 FA2A01     319 +         JP   M,B_0001      ;IF SUBTRACTION NEGATIVE JUMP
0127 215821     319 +         LD   HL,NWTXSQ     ;HL=ADDRESS OF START OF QUEUE
012A 225421     319 +B_0001   LD   (NWTXHQ),HL   ;STORE VALUE OF HL INTO TAIL ADDRESS
012D 215321     320           LD   HL,NTXOFR     ;ADDRESS OF NETWORK QUEUE STATUS WORD
0130 CB86       321           RES  0,(HL)        ;SET STATUS WORD TO PROCESSOR #2
0132 CB96       322           RES  2,(HL)        ;NOT WAITING OR USING
0134 FB         323           EI                 ;ENABLE INTERRUPT
                324  :****THIS SECTION OF CODE TRANSMITS THE MESSAGE WHOSE MEMORY
                325  :BLOCK STORAGE ADDRESS WAS AT THE HEAD OF NETWORK TX QUEUE.
                326  :A SUBROUTINE (NWTXSR) IS USED TO TRANSMIT THE MESSAGE.
                327  :THIS SECTION ASSUMES THE MESSAGE IS STORED IN THE
                328  :MEMORY BLOCK IN THE FOLLOWING SEQUENCE
                329  :START OF BLOCK   MESSAGE LENGTH
                330  :                 +1-BLANK
                331  :                 +2-DESTINATION ADDRESS
                332  :                 +3-SENDER ADDRESS
                333  :                 +4-FIRST WORD IN MESSAGE
0135 03         334           INC  BC            ;BC=ADDRESS OF SECOND WORD
0136 03         335           INC  BC            ;IN MEMORY BLOCK
0137 0A         336           LD   A,(BC)        ;A=DESTINATION ADDRESS
0138 0B         337           DEC  BC            ;STORE THE DESTINATION ADDRESS
0139 02         338           LD   (BC),A        ;IN FIRST WORD OF MEMORY BLOCK
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

013A 21D610     339           LD   HL,MSGID      ;HL=ADD OF LINK MSG ID WORD
013D 34         340           INC  (HL)          ;INCREMENT LINK MSG ID WORD
013E 7E         341           LD   A,(HL)        ;A=LINK MSG ID WORD
013F 03         342           INC  BC            ;STORE THE LINK MSG ID WORD
0140 02         343           LD   (BC),A        ;IN THE SECOND WORD OF MEMORY BLOCK
0141 0B         344           DEC  BC            ;BC=MEMORY BLOCK STORAGE ADDRESS
0142 0B         345           DEC  BC            ;OF MESSAGE TO BE TRANSMITTED
0143 CDC502     346           CALL NWTXSR        ;TRANSMIT MESSAGE
                347  ;****THIS SECTION OF CODE PUT THE JUST TRANSMITTED MESSAGE
                348  ;AT THE END OF THE ALREADY TRANSMITTED QUEUE. A CHECK IS
                349  ;MADE TO DETERMINE IF THE HEAD ADDRESS IS AT THE END OF
                350  ;THE QUEUE. IF SO IT IS REINITIALIZED TO START OF QUEUE ADDRESS
0146            351  NMAN1A  ADDTQ LOTXSQ,LOTXEQ,LOTXTQ
0146 2A4D20     351 +         LD   HL,(LOTXTQ)   ;HL=ADDRESS OF TAIL OF QUEUE
0149 71         351 +         LD   (HL),C        ;PUT LOW ORDER BYTE OF THE MESSAGE
                351 +;                            MEMORY BLOCK AT TAIL OF QUEUE
014A 23         351 +         INC  HL            ;PUT HIGH ORDER BYTE OF THE MESSAGE
014B 70         351 +         LD   (HL),B        ;MEMORY BLOCK AT TAIL OF QUEUE
014C 23         351 +         INC  HL            ;HL=NEW TAIL OF QUEUE
014D E5         351 +         PUSH HL            ;SAVE NEW TAIL OF QUEUE ADDRESS
014E 115221     351 +         LD   DE,LOTXEQ     ;DE=ADDRESS OF END OF QUEUE
0151 ED52       351 +         SBC  HL,DE         ;HL=CURRENT LOCATION-END QUEUE-CARRY
0153 E1         351 +         POP  HL            ;HL=ADD OF CURRENT TAIL OF QUEUE
0154 FA5A01     351 +         JP   M,A_0002      ;IF SUBTRACTION NEGATIVE JUMP
0157 214F20     351 +         LD   HL,LOTXSQ     ;HL=ADDRESS OF START OF QUEUE
015A 224D20     351 +A_0002   LD   (LOTXTQ),HL   ;STORE VALUE OF HL INTO TAIL ADDRESS
                352  ;****THIS SECTION OF CODE STARTS THE NETWORK MESSAGE TIMER.******
                353  ;THIS TIMER IS USED TO GENERATE AN INTERRUPT IF CONFORMATION
                354  ;OF A TRANSMITTED MESSAGE IS NOT RECEIVED WITHIN A USER
                355  ;SPECIFIED TIME PERIOD
015D 3E35       356  NMAN1B  LD   A,00110101B    ;RESET THE Z80 CTC
015F D3FA       357           OUT  (250D),A      ;CHANNEL 2
0161 3EFF       358           LD   A,11111111B
0163 D3FA       359           OUT  (250D),A
0165 3EB7       360           LD   A,10110111B   ;RESET THE Z80-CTC
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

0167 D3F8       361           OUT   (2510),A     ;CHANNEL 3 TIMER
0169 3EFF       362           LD    A,11111111B  ;ASSOCIATED WITH THE
016B D3F8       363           OUT   (2510),A     ;MESSAGE TIMER FUNCTION
016D C3F700     364           JP    NMAN02       ;JUMP TO MONITOR OF NW RECEIVE QUEUE
                365  ;****THIS SECTION OF CODE OBTAINS THE MEMORY BLOCK STORAGE ******
                366  ;ADDRESS OF THE MESSAGE AT THE HEAD OF THE NETWORK RECEIVE QUEUE
                367  ;A CHECK IS ALSO MADE TO DETERMINE IF THE HEAD ADDRESS OF QUEUE
                368  ;IS AT THE END OF THE QUEUE. IF SO IT IS REINITIALIZED TO START
                369  ;OF QUEUE ADDRESS. THIS SECTION ASSUMES THE MESSAGE STORED
                370  ;IN THE MEMORY BLOCK IS IN THE FOLLOWING SEQUENCE
                371  ;START OF BLOCK  -MESSAGE LENGTH
                372  ;                +1-DESTINATION ADDRESS
                373  ;                +2-LINK CONTROL MESSAGE ID
                374  ;                +3-SENDER ADDRESS
                375  ;                +4-FIRST WORD IN MESSAGE
0170            376  DNWRXQ  SUBHQ NWRXSQ,NWRXEQ,NWRXTQ
0170 2A5210     376 +        LD    HL,(NWRXTQ)  ;HL=ADDRESS OF HEAD OF QUEUE
0173 4E         376 +        LD    C,(HL)       ;PUT LOW ORDER BYTE OF ADDRESS AT
                376 +;                          HEAD OF QUEUE INTO REG C
0174 23         376 +        INC   HL           ;PUT HIGH ORDER BYTE OF ADDRESS AT HEAD
0175 46         376 +        LD    B,(HL)       ;QUEUE INTO REG B
0176 23         376 +        INC   HL           ;HL=NEW HEAD OF QUEUE ADDRESS
0177 E5         376 +        PUSH  HL           ;SAVE NEW HEAD OF QUEUE ADDRESS
0178 ED52       376 +        SBC   HL,DE        ;HL=CURRENT LOCATION-END QUEUE-CARRY
017A E1         376 +        POP   HL           ;HL=ADD OF CURRENT TAIL OF QUEUE
017B FA8101     376 +        JP    M,B_0003     ;IF SUBTRACTION NEGATIVE JUMP
017E 215410     376 +        LD    HL,NWRXSQ    ;HL=ADDRESS OF START OF QUEUE
0181 225210     376 +B_0003  LD    (NWRXTQ),HL  ;STORE VALUE OF HL INTO TAIL ADDRESS
0184 C5         377  NMAN2A  PUSH  BC           ;SET THE IX REGISTER EQUAL
0185 DDE1       378          POP   IX           ;TO THE MEMORY BLOCK ADDRESS
0187 DD7E01     379          LD    A,(IX+1)     ;A=TO ADDRESS OF MESSAGE
                380  ;                           ---NOTE THIS OFFSET WOULD DEPEND
                381  ;                           ON MESSAGE STRUCTURE. IN THIS
                382  ;                           CASE ASSUMED TO BE FIRST WORD
                383  ;                           IN MESSAGE---
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND      COMMENT

018A FE80       384           CP   1280         ;COMPARE IT TO NETWORK INTERFACE ADDRES
                385  :                          ---NOTE NETWORK ADDRESS OF
                386  :                          INTERFACE WOULD BE DEPENDENT
                387  :                          UPON NETWORK. IN THIS CASE
                388  :                          ASSUMED TO BE 1280
018C CAD301     389           JP   Z,CONMSG     ;IF ZERO HAVE CONTROL MESSAGE SO JUMP
                390  ;****THIS SECTION OF CODE DETERMINES IF THE MESSAGE        ******
                391  ;IS ADDRESSED TO A LOCAL PERIPHERAL. THIS IS DONE THROUGH
                392  ;A LINEAR SEARCH OF THE NETWORK ADDRESS TABLE
018F 214920     393           LD   HL,NWTBNU    ;ADD NUMBER OF ENTRIES IN TABLE
0192 5E         394           LD   E,(HL)       ;NUMBER OF ENTRIES IN THE NETWORK TABLE
0193 214720     395           LD   HL,NWADTB    ;HL=ADDRESS OF NETWORK ADDRESS TABLE
0196 BE         396  NWSCA1   CP   (HL)         ;COMPARE TO ADDRESS OF MESSAGE TO FIRST
                397  :                          IN THE TABLE
0197 CAA201     398           JP   Z,ADDMT1     ;IF ZERO HAVE ADDRESS MATCH SO JUMP
019A 1D         399           DEC  E            ;E=E - 1
019B CACA01     400           JP   Z,NOTMT1     ;IF ZERO TO ADDRESS DOES NOT MATCH ANY
                401  :                          NETWORK ADDRESS TABLE SO JUMP
019E 23         402           INC  HL           ;ADD OF NEXT ENTRY IN NETWORK TABLE
019F C39601     403           JP   NWSCA1       ;REPEAT THE LOOP
                404  ;****THIS SECTION OF CODE IS ENTERED IF THE MESSAGE
                405  ;WAS ADDRESSED TO A LOCAL PERIPHERAL. THE MESSAGE MEMORY
                406  ;BLOCK STORAGE ADDRESS IS PUT ON THE LOCAL TX QUEUE AND
                407  ;AN ACKNOWLEDGEMENT MESSAGE IS CONSTRUCTED AND TRANSMITTED.
01A2 6F         408  ADDMT1   LD   L,A          ;L=DESTINATION ADDRESS
01A3 DD7E02     409           LD   A,(IX+2)     ;A=LINK CONTROL MESSAGE ID
01A6 F5         410           PUSH AF           ;SAVE MESSAGE IDENTIFICATION
01A7 7D         411           LD   A,L          ;A=DESTINATION ADDRESS
01A8 DD7702     412           LD   (IX+2),A     ;OVERWRITE MESSAGE ID WORD
01AB DD3500     413           DEC  (IX+0)       ;AND REDUCE MESSAGE LENGTH BY ONE
01AE CD4D03     414           CALL LOTXQN       ;PUT MESSAGE ON LOCAL TRANSMIT QUEUE
01B1 CDEE02     415           CALL ALNMEN       ;GET FREE STORAGE BLOCK
01B4 C5         416           PUSH BC           ;SAVE NEW BLOCK STORAGE ADDRESS
01B5 3E03       417           LD   A,03D        ;A=NUMBER OF WORDS IN ACK MESSAGE
                418  :                          ---NOTE THIS NUMBER WOULD BE SET
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

               419   :                          BY LINK PROTOCOL IN USE. IN
               420   :                          THIS CASE ASSUMED TO BE 3
01B7 02        421           LD   (BC),A        :BC=MESSAGE LENGTH OF 3
               422   :                          ---THIS WOULD BE NETWORK DEPENDENT
01B8 03        423           INC  BC            :BC=NEXT ADDRESS OF MESSAGE
01B9 3E81      424           LD   A,129         :A=ADDRESS OF CONNECTING NETWORK DEVICE
01BB 02        425           LD   (BC),A        :STORE THE ADDRESS
01BC 03        426           INC  BC            :BC=NEXT STORAGE LOCATION
01BD 3E80      427           LD   A,128D        :A=LOCAL INTERFACE I/O ADDRESS
01BF 02        428           LD   (BC),A        :STORE LOCAL INTERFACE I/O ADDRESS
01C0 C1        429           POP  BC            :BC=MEMORY BLOCK STORAGE ADDRESS
01C1 CDC502    430           CALL NWTXSR        :TRANSMIT THE ACKNOWLEDGEMENT MESSAGE
01C4 CD2E03    431           CALL DANMEN        :RETURN THE ACK MESSAGE MEMORY ADDRESS
01C7 C3EA00    432           JP   NMAN01        :JUMP TO MONITOR NETWORK TX QUEUE
               433   :****THIS SECTION OF CODE IS ENTERED IF THE MESSAGE WAS    ******
               434   :NOT ADDRESSED TO A LOCAL PERIPHERAL. THE MEMORY BLOCK IS
               435   :RESTRUCTURE TO MATCH THAT FOR A MESSAGE ON THE NETWORK
               436   :TX QUEUE AND PUT ON THAT QUEUE.
01CA DD7702    437   NOTMT1 LD   (IX+2),A     :OVERWRITE MESSAGE ID WORD
01CD CD7D03    438           CALL NWTXQN        :PUT MESSAGE ON NETWORK TRANSMIT QUEUE
01D0 C3EA00    439           JP   NMAN01        :JUMP TO MONITOR NETWORK TX QUEUE
               440   :****THIS SECTION IS EXECUTED IF THE RECEIVED MESSAGE      ******
               441   :WAS A MESSAGE ACKNOWLEDGEMENT. THE MESSAGE BEING
               442   :ACKNOWLEDGE IS COMPARE WITH THE MESSAGE IDENTIFICATION
               443   :OF THE MESSAGE ON THE ALREADY TRANSMITTED QUEUE
01D3 C5        444   CONMSG PUSH BC            :BC=MEMORY BLOCK ADD OF RECEIVED MSG
01D4           445   CONMG1 SUBHD NATXSQ,NATXEQ,NATXHQ
01D4 2A0A10    445 +         LD   HL,(NATXHQ)   :HL=ADDRESS OF HEAD OF QUEUE
01D7 4E        445 +         LD   C,(HL)        :PUT LOW ORDER BYTE OF ADDRESS AT
               445 +:                           HEAD OF QUEUE INTO REG C
01D8 23        445 +         INC  HL            :PUT HIGH ORDER BYTE OF ADDRESS AT HEAD
01D9 46        445 +         LD   B,(HL)        :QUEUE INTO REG B
01DA 23        445 +         INC  HL            :HL=NEW HEAD OF QUEUE ADDRESS
01DB E5        445 +         PUSH HL            :SAVE NEW HEAD OF QUEUE ADDRESS
01DC ED52      445 +         SBC  HL,DE         :HL=CURRENT LOCATION-END QUEUE-CARRY
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

01DE E1         445 +          POP  HL           ;HL=ADD OF CURRENT TAIL OF QUEUE
01DF FAE501     445 +          JP   M,B_0004     ;IF SUBTRACTION NEGATIVE JUMP
01E2 210E10     445 +          LD   HL,NATXSQ    ;HL=ADDRESS OF START OF QUEUE
01E5 220A10     445 +B_0004    LD   (NATXHQ),HL  ;STORE VALUE OF HL INTO TAIL ADDRESS
01E8 C5         446  CONMG3    PUSH BC           ;BC=MEMORY BLOCK ADDRESS OF MSG AT HEAD
                447  :                           ALREADY TRANSMIT QUEUE
01E9 03         448            INC  BC           ;BC=LOCATION OF MESSAGE
01EA 03         449            INC  BC           ;IDENTIFICATION
01EB 0A         450            LD   A,(BC)       ;A=MESSAGE ID
01EC DD9E02     451            CP   (IX+2)       ;COMPARE A TO LOCATION OF ACK MSG ID
                452  :                           ---NOTE THIS OFFSET WOULD BE
                453  :                           DICTATED BY PROTOCOL IN USE
                454  :                           IN THIS CASE ASSUMED TWO---
01EF CA0A02     455            JP   Z,CONMG2     ;IF ZERO PROPER ACK SO JUMP
01F2 C1         456            POP  BC           ;BC=HEAD OF ALREDY TRANSMITTED QUEUE
01F3 CD7D03     457            CALL NWTXQN       ;PUT MSG ON QUEUE FOR RETRANSMISSION
01F6 2A0A10     458            LD   HL,(NATXHQ)  ;HL=ALREADY TX HEAD OF QUEUE
01F9 ED5B0C10   459            LD   DE,(NATXTQ)  ;DE=ALREADY TX TAIL OF QUEUE
01FD AF         460            XOR  A            ;CARRY=0
01FE ED52       461            SBC  HL,DE        ;HL=ALREADY TX HEAD - ALREADY TX TAIL-0
0200 C20401     462            JP   NZ,CONMG1
0203 C1         463            POP  BC           ;MEMORY BLOCK ADD RECEIVED MESSAGE
0204 CD2E03     464            CALL DANMEN       ;FREE THE MEMORY BLOCK
0207 C3EA00     465            JP   NMAN01       ;JUMP TO MONITOR NETWORK TX QUEUE
020A C1         466  CONMG2    POP  BC           ;BC=ADDRESS OF MESSAGE AT HEAD ALREADY
020B CD2E03     467            CALL DANMEN       ;FREE THE MEMORY BLOCK
020E DDE5       468            PUSH IX           ;SAVE RECEIVED MESSAGE MEMORY BLOCK ADD
0210 C1         469            POP  BC           ;BC=ADDRESS OF MESSAGE RECEIVED
0211 CD2E03     470            CALL DANMEN       ;FREE THE MEMORY BLOCK
0214 3E35       471            LD   A,00110101B  ;RESET THE Z80 CTC
0216 D3FA       472            OUT  (250D),A     ;CHANNEL 2
0218 3EFF       473            LD   A,11111111B
021A D3FA       474            OUT  (2300),A
021C 3EB7       475            LD   A,10110111B  ;RESET THE Z80-CTC
021E D3FB       476            OUT  (2510),A     ;CHANNEL 3 TIMER
```


0000
0002
0004

0041
0044
0045
0047
004B
004D

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

0220 3EFF       477          LD   A,11111111B  :ASSOCIATED WITH THE
0222 D3FB       478          OUT  (251D),A     :MESSAGE TIMER FUNCTION
0224 C3EA00     479          JP   NMAN01       :JUMP TO MONITOR NETWORK TX QUEUE
                480 :*****************************************************************
                481 :**************    SUBROUTINE SPECIAL RECEIVE    ****************
                482 :*****************************************************************
                483 :THIS SUBROUTINE REPRESENTS A TYPICAL INTERRUPT SERVICE ROUTINE
                484 :FOR HANDLING A SPECIAL RECEIVE CONDITION INTERRUPT. A SPECIAL
                485 :RECEIVE CONDITION INTERRUPT IS GENERATED IN RESPONSE TO A PARITY
                486 :ERROR, RECEIVER OVERRUN ERROR, A CRC/FRAMING ERROR OR AN END OF
                487 :FRAME CHARACTER (01111110). THE SERVICE ROUTINE FIRST DETERMINES
                488 :THE CAUSE OF THE INTERRUPT BY READING REGISTER ONE OF THE Z80A-
                489 :SIO AND TESTING THE APPROPRIATE BIT. IF IT INTERRUPT WAS CAUSED
                490 :BY AN ERROR, THE ROUTINE IS EXITED TO AWAIT AN END OF MESSAGE
                491 :INTERRUPT. IF AN END OF MESSAGE INTERRUPT IS ENCOUNTERED THE
                492 :ROUTINE DETERMINES IF THE TRANSMITTER WAS ACTIVE WHEN THE
                493 :INTERRUPT OCCURRED. IF SO THE SUBROUTINE COMPLETES TRANSMISSION
                494 :OF THE MESSAGE BEFORE PROCESSING THE NETWORK RECEIVED MESSAGE.
                495 :THIS LATTER MESSAGE IS PROCESSED BY FIRST DETERMINING IF THE
                496 :MESSAGE HAD A PREVIOUS ERROR. IF SO THE BLOCK STORAGE  ADDRESS
                497 :IS REINITIALIZED ALONG WITH THE ALTERNATE REGISTER SET IN AN-
                498 :TICIPATION OF THE NEXT NETWORK MESSAGE. IF THE MESSAGE WAS
                499 :ERROR FREE, IT IS PLACED ON THE RECEIVE NETWORK QUEUE FOR
                500 :PROCESSING BY THE MAIN ROUTINE
                501 :                        IMPORTANT LABELS
                502 :    NWRXQ -A SECTION OF CODE TO PUT RECEIVED MESSAGE MEMORY
                503 :           BLOCK ADDRESS ONTO RECEIVED NETWORK QUEUE
                504 :    NWTXSR-A SUBROUTINE USED TO TRANSMIT A NETWORK MESSAGE
                505 :                          REGISTERS
                506 :THIS SUBROUTINE ASSUMES THE ALTERNATIVE REGISTER SET IS
                507 :DEDICATED TO THE RECEIVE NETWORK FUNCTION. THE REGISTER
                508 :CONTENT IS AS FOLLOWS:
                509 :    BC -MEMORY BLOCK STORAGE START ADDRESS
                510 :    HL -CURRENT STORAGE LOCATION
                511 :    E  -MESSAGE COUNT
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

               512   :                          SPECIAL INST.
               513   :TO USE THIS ROUTINE, THE NETWORK MESSAGE MUST BE DELIMITED
               514   :BY FLAGS(01111110)
0227 08        515   SXA01   EX   AF,AF' :SAVE INTERRUPTED PROGRAM A REGISTER AND FLAG
0228 3E01      516           LD   A,00000001B  :SET PORT A SIO
022A D3FE      517           OUT  (254D),A     :TO REGISTER 1
               518   :                          ---NOTE I/O PORT ADDRESS OF SIO
               519   :                          WOULD BE USER SPECIFIED---
022C DBFE      520           IN   A,(254D)     :A=CONTENTS OF REGISTER 1
022E CB7F      521           BIT  7,A          :CHECK IF END OF MESSAGE
0230 CA7602    522           JP   Z,WDERR1     :IF ZERO WORD ERROR SO JUMP
0233 F5        523           PUSH AF           :SAVE RECEIVED MESSAGE'S STATUS
0234 DBFE      524           IN   A,(254D)     :READ PORT A REGISTER 0. NOTE REGISTER
               525   :                          ADDRESS NOT REQUIRED SINCE
               526   :                          REGISTER ADDRESS RESET TO 0
               527   :                          AFTER READ FROM REGISTER 1
0236 CB57      528           BIT  2,A          :CHECK IF TRANSMIT IS ENABLED
0238 CA4702    529           JP   Z,NSA01      :IF ZERO TRANSMITTER NOT ENABLE SO JUMP
023B CDE602    530           CALL NTXSR1       :CALL ROUTINE TO COMPLETE TRANSMISSION
               531   :                          OF THE MESSAGE
023E F1        532           POP  AF           :RESTORE RECEIVED MESSAGE STATUS
023F E1        533           POP  HL           :HL=INVALID RETURN ADDRESS SINCE MSG TX
0240 21ED02    534           LD   HL,NTXSR2    :NEW VALID RETURN ADDRESS
0243 E5        535           PUSH HL           :PUT NEW RETURN ADDRESS ON STACK
0244 C34802    536           JP   NSA02        :JUMP OVER NEXT PART
0247 F1        537   NSA01   POP  AF           :RESTORE RECEIVED MESSAGE STATUS
0248 D9        538   NSA02   EXX               :SAVE INTERRUPTED PROGRAM REGISTER SET
0249 CB77      539           BIT  6,A          :HAVE EOM, CHECK IF MSG WAS ERROR FREE
024B CA6C02    540           JP   Z,RXERR1     :IF ZERO MESSAGE ERROR SO JUMP
024E 1D        541           DEC  E            :REDUCED MESSAGE LENGTH
024F 1D        542           DEC  E            :BY THE TWO ERROR CONTROL WORDS
0250 7B        543           LD   A,E          :STORE MESSAGE LENGTH AT MESSAGE
0251 02        544           LD   (BC),A       :LENGTH WORD ADDRESS
0252           545           ADDTQ NWRXSQ,NWRXEQ,NWRXTQ
0252 2A5210    545 +         LD   HL,(NWRXTQ)  :HL=ADDRESS OF TAIL OF QUEUE
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

0255 71         545 +         LD   (HL),C       ;PUT LOW ORDER BYTE OF THE MESSAGE
                545 +;                           MEMORY BLOCK AT TAIL OF QUEUE
0256 23         545 +         INC  HL           ;PUT HIGH ORDER BYTE OF THE MESSAGE
0257 70         545 +         LD   (HL),B       ;MEMORY BLOCK AT TAIL OF QUEUE
0258 23         545 +         INC  HL           ;HL=NEW TAIL OF QUEUE
0259 E5         545 +         PUSH HL           ;SAVE NEW TAIL OF QUEUE ADDRESS
025A 11D510     545 +         LD   DE,NWRXEQ    ;DE=ADDRESS OF END OF QUEUE
025D ED52       545 +         SBC  HL,DE        ;HL=CURRENT LOCATION-END QUEUE-CARRY
025F E1         545 +         POP  HL           ;HL=ADD OF CURRENT TAIL OF QUEUE
0260 FA6602     545 +         JP   M,A_0005     ;IF SUBTRACTION NEGATIVE JUMP
0263 215410     545 +         LD   HL,NWRXSQ    ;HL=ADDRESS OF START OF QUEUE
0266 225210     545 +A_0005   LD   (NWRXTQ),HL  ;STORE VALUE OF HL INTO TAIL ADDRESS
0269 CDEE02     546  NSA03    CALL ALNMEM       ;GET NEW MEMORY BLOCK ADDRESS
026C 60         547  RXERR1   LD   H,B          ;HL=MEMORY BLOCK STORAGE
026D 69         548           LD   L,C          ;   START ADDRESS
026E 23         549           INC  HL           ;FIRST AVAILABLE BLOCK STORAGE ADD
026F 1E00       550           LD   E,0          ;E=MESSAGE LENGTH = 2
0271 3E70       551           LD   A,01110000B  ;RESET CRC CHECKER
0273 D3FE       552           OUT  (254Q),A     ;AND ERROR LATCH
0275 D9         553           EXX               ;REGISTER SET
0276 08         554  WDERR1   EX   AF,AF' :RESTORE REG A
0277 FB         555           EI                ;ENABLE INTERRUPTS
0278 ED4D       556           RETI              ;RETURN FROM INTERRUPT
                557 ;*************************************************************
                558 ;-************    SUBROUTINE NETWORK RECEIVE    ***************
                559 ;*************************************************************
                560 ;THIS SUBROUTINE REPRESENTS A TYPICAL INTERRUPT SERVICE ROUTINE
                561 ;FOR RECEIPT OF A NETWORK MESSAGE. THIS ROUTINE ASSUMES THE
                562 ;ALTERNATE REGISTER SET IS DEDICATED TO THE NETWORK MESSAGE
                563 ;RECEIVE FUNCTION. THE REGISTERS ARE USED AS FOLLOWS:BC=MEMORY
                564 ;BLOCK STORAGE START ADDRESS, HL=CURRENT STORAGE ADDRESS, E=
                565 ;MESSAGE LENGTH A = INPUT REGISTER
027A 08         566  RXA01    EX   AF,AF' :SAVE THE A REG
027B D9         567           EXX               ;SAVE REGISTERS BC,HL,DE
                568 ;THIS ALSO RESTORES THE DEDICATED NETWORK REGISTER SET
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

027C DBFC       569          IN   A,(2520)      ;INPUT THE NETWORK MESSAGE WORD
                570  ;                          ---NOTE I/O PORT ADDRESS OF SIO
                571  ;                          WOULD BE USER SPECIFIED---
027E 77         572          LD   (HL),A        ;STORE THE NETWORK WORD
027F 23         573          INC  HL            ;HL=NEXT STORAGE ADDRESS
0280 1C         574          INC  E             ;INCREMENT THE MESSAGE LENGTH
0281 08         575          EX   AF,AF' ;RESTORE THE A REG
0282 D9         576          EXX                ;RESTORE THE INTERRUPTED PROGRAM REGIST
0283 FB         577          EI                 ;ENABLE INTERRUPTS
0284 ED4D       578          RETI               ;RETURN FROM INTERRUPT
                579  ;*************************************** ***************
                580  ;**************          SUBROUTINE TIMER         ***************
                581  ;*************************************** ***************
                582  ;THIS SUBROUTINE REPRESENTS A TYPICAL INTERRUPT SERVICE ROUTINE
                583  ;FOR HANDLING A MESSAGE TIMEOUT INTERRUPT. THE TIMER IS USED
                584  ;TO GENERATE AN INTERRUPT AFTER A USER SPECIFIED AMOUNT OF TIME.
                585  ;THE TIMER IS NORMALLY ACTIVATED UPON TRANSMISSION OF A MESSAGE.
                586  ;AFTER THE SPECIFIED TIME HAS ELAPSED AN INTERRUPT IS GENERATED
                587  ;CAUSING THE MESSAGE TO BE PUT ON THE NETWORK TRANSMIT QUEUE.
                588  ;A CHECK IS MADE TO INSURE A MESSAGE IS ON THE ALREADY TRANS-
                589  ;MITTED QUEUE. THE INTERRUPTED PROGRAM REGISTERS ARE SAVED
                590  ;THROUGH THE PUSH INSTRUCTION SINCE THE ALTERNATE REGISTER
                591  ;SET IS DEDICATED TO THE NETWORK RECEIVE FUNCTION.
0286 F5         592  TIME01  PUSH AF            ;SAVE INTERRUPTED
0287 C5         593          PUSH BC            ;PROGAM REGISTER
0288 E5         594          PUSH HL            ;SET
0289 D5         595          PUSH DE            ;
028A FB         596          EI                 ;ENABLE INTERRUPT
028B 2A0A10     597          LD   HL,(NATXHQ)   ;HL=HEAD OF ALREADY TX QUEUE
028E ED5B0C10   598          LD   DE,(NATXTQ)   ;DE=TAIL OF ALREADY TX QUEUE
0292 AF         599          XOR  A             ;A=0
0293 ED52       600          SBC  HL,DE         ;HL=HEAD - TAIL
0295 CABF02     601          JP   Z,TIME11      ;JUMP IF QUEUE EMPTY
0298            602          SUBHQ NATXSQ,NATXEQ,NATXHQ
0298 2A0A10     602 +        LD   HL,(NATXHQ)   ;HL=ADDRESS OF HEAD OF QUEUE
```


00C4
00C7
00CA
00CD
00D0
00D3
00D6
00D9
00DC

```
ADDR OBJECT    STMT  LABEL    OPCO OPERAND       COMMENT

029B 4E         602 +         LD   C,(HL)       :PUT LOW ORDER BYTE OF ADDRESS AT
                602 +:                           HEAD OF QUEUE INTO REG C
029C 23         602 +         INC  HL           :PUT HIGH ORDER BYTE OF ADDRESS AT HEAD
029D 46         602 +         LD   B,(HL)       ;QUEUE INTO REG B
029E 23         602 +         INC  HL           :HL=NEW HEAD OF QUEUE ADDRESS
029F E5         602 +         PUSH HL           :SAVE NEW HEAD OF QUEUE ADDRESS
02A0 ED52       602 +         SBC  HL,DE        :HL=CURRENT LOCATION-END QUEUE-CARRY
02A2 E1         602 +         POP  HL           :HL=ADD OF CURRENT TAIL OF QUEUE
02A3 FAA902     602 +         JP   M,$_0006     :IF SUBTRACTION NEGATIVE JUMP
02A6 210E10     602 +         LD   HL,NATXSQ    :HL=ADDRESS OF START OF QUEUE
02A9 220A10     602 +$_0006   LD   (NATXHQ),HL  :STORE VALUE OF HL INTO TAIL ADDRESS
02AC CD7D03     603           CALL NWTXQN       :PUT THIS ADDRESS AT END OF NETWORK TX
02AF 3E35       604           LD   A,00110101B  :RESET THE Z80 CTC
02B1 D3FA       605           OUT  (250D),A     :CHANNEL 2
02B3 3EFF       606           LD   A,11111111B
02B5 D3FA       607           OUT  (250D),A
02B7 3E87       608           LD   A,10110111B  :RESET THE Z80-CTC
02B9 D3FB       609           OUT  (251D),A     :CHANNEL 3 TIMER
02BB 3EFF       610           LD   A,11111111B  :ASSOCIATED WITH THE
02BD D3FB       611           OUT  (251D),A     :MESSAGE TIMER FUNCTION
02BF D1         612 TIME11    POP  DE           :RESTORE THE
02C0 E1         613           POP  HL           :INTERRUPTED
02C1 C1         614           POP  BC           :PROGRAM
02C2 F1         615           POP  AF           :REGISTER SET
02C3 ED4D       616           RETI              :RETURN FROM INTERRUPT
                617 :*************************************************************
                618 :**************    SUBROUTINE NETWORK TRANSMIT   ****************
                619 :*************************************************************
                620 :THIS SUBROUTINE REPRESENTS A TYPICAL SERVICE ROUTINE FOR
                621 :TRANSMISSION OF A NETWORK MESSAGE. THE SERVICE ROUTINE IS
                622 :USED IN CONJUNCTION WITH THE Z80A-SIO TO TRANSMIT THE MESSAGE.
                623 :THE FIRST PART OF THE ROUTINE INITIALIZES THE TRANSMIT
                624 :PORT OF THE SIO. AFTER THE SIO IS INITIALIZES A LOOP IS ENTER
                625 :WHICH TRANSMIT THE MESSAGE.
02C5 C5         626 NWTXSR    PUSH BC           :SAVE THE MEMORY BLOCK STORAGE ADDRESS
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

02C6 69         627          LD   L,C           ;HL=THE MEMORY BLOCK
02C7 60         628          LD   H,B           STORAGE ADDRESS
02C8 46         629          LD   B,(HL)        ;B=MEMORY BLOCK STORAGE LENGTH
02C9 23         630          INC  HL            ;HL=ADDRESS OF START OF MESSAGE
02CA 0EFC       631          LD   C,252D        ;C=I/O PORT ADDRESS OF PORT A SIO
                632  ;                          ---I/O ADDRESS IS USER SPECIFIED
02CC 3E80       633          LD   A,10000000B   ;RESET PORT A TRANSMIT
02CE D3FE       634          OUT  (254D),A      ;CRC GENERATOR
02D0 3E06       635          LD   A,00000110B   ;SET PORT A COMMAND
02D2 D3FE       636          OUT  (254D),A      ;REGISTER ADDRESS TO 5
02D4 3EEF       637          LD   A,11101111B   ;ENABLE PORT A TRANSMITTER
02D6 D3FE       638          OUT  (254D),A      ;AND CRC GENERATOR
02D8 7E         639          LD   A,(HL)        ;LOAD FIRST WORD OF MESSAGE
02D9 EDA3       640          OUTI               ;INTO A AND OUTPUT THE WORD
02DB 3E00       641          LD   A,0D          ;SET PORT A COMMAND
02DD D3FE       642          OUT  (254D),A      ;REGISTER ADDRESS TO 0
02DF DBFE       643  NWTX01  IN   A,(254D)      ;READ PORT A SIO STATUS REGISTER
02E1 CB57       644          BIT  2,A           ;DETERMINE IF TX BUFFER EMPTY
02E3 CADF02     645          JP   Z,NWTX01
02E6 7E         646  NTXSR1  LD   A,(HL)        ;A=NEXT WORD
02E7 EDA3       647          OUTI               ;OUTPUT NEXT WORD
02E9 C2DF02     648          JP   NZ,NWTX01     ;IF B NOT EQUAL TO ZERO REPEAT THE LOOP
02EC C1         649          POP  BC            ;BC=MEMORY BLOCK ADDRESS OF MSG JUST TX
02ED C9         650  NTXSR2  RET                ;RETURN TO CALLING PROGRAM
                651  ;*************************************************************
                652  ;**************SUBROUTINE ALLOCATE MEMORY NETWORK*****************
                653  ;*************************************************************
                654  ;THIS SUBROUTINE ALLOCATES A BLOCK OF MEMORY FOR MESSAGE
                655  ;STORAGE. A MEMORY TABLE (LOMNTB) IS USED TO DETERMINE
                656  ;WHERE STORAGE IS AVAILABLE. THIS MEMORY TABLE CONTAINS
                657  ;THE START ADDRESS OF ALL UNALLOCATED MEMORY BLOCKS. A
                658  ;CHECK IS MADE TO DETERMINE IF A MEMORY BLOCK IS AVAIL-
                ;ABLE. IF NOT, THE SUBROUTINE ENTERS A WAIT LOOP (ALGONN).
```

```
ADDR OBJECT    STMT  LABEL   OPCD OPERAND       COMMENT

               662  :THE ALLOCATED BLOCK ADDRESS IS RETURNED IN THE BC REGISTERS
               663  :
               664  :                              LABELS
               665  :ALNMEN- NAME OF SUBROUTINE
               666  :ALGONN- START OF WAIT LOOP IF MEMORY UNAVAILABLE
               667  :MNTBED- ADDRESS OF LAST ENTRY IN MEMORY TABLE
               668  :MNTBFR- PROVIDES USAGE STATUS OF MEMORY TABLE
               669  :MNTBPT- MEMORY TABLE ADDRESS OF NEXT FREE MEMORY BLOCK
               670  :
               671  :                            REGISTERS
               672  :THE SUBROUTINE DOES NOT SAVE ANY REGISTERS FROM THE CALL-
               673  :ING PROGRAM. THE SUBROUTINE USES REGISTERS BC,HL,DE. THE
               674  :BLOCK ADDRESS IS RETURNED IN REGISTER BC
               675  :
               676  :                          SPECIAL INST
               677  :SINCE THIS SUBROUTINE CAN PUT THE OTHER PROCESSOR INTO A
               678  :WAIT LOOP, THE CALLING ROUTINE SHOULD DISABLE INTERRUTS
               679  :BEFORE CALLING THE SUBROUTINE. IF THE ALGONE PORTION OF
               680  :THE SUBROUTINE IS EXECUTED THE ROUTINE DISABLES INTERRUPTS
02EE 21401F    681  ALNMEN  LD   HL,MNTBFR    :LOCATION OF MEMORY TABLE STATUS WORD
02F1 CBD6      682         SET  2,(HL)        :STATUS WORD TO PROCESSOR #2 WAITING
02F3 CB4E      683         BIT  1,(HL)        :CHECK IF PROCESSOR #1 WAITING
02F5 CAFD02    684         JP   Z,ALNME1      ;JUMP IF PROCESSOR #1 NOT WAITING
02F8 CB46      685  ALNME2  BIT  0,(HL)        ;IF PROCESSOR #1 ALSO WAITING CHECK
               686  :                           TO SEE IF PROCESSOR #1 USING
02FA C2F802    687         JP   NZ,ALNME2     :LOOP IF PROCESSOR #1 USING
02FD CBC5      688  ALNME1  SET  0,(HL)        :SET STATUS WORD TO PROCESSOR #2 USING
02FF 2A411F    689         LD   HL,(MNTBPT)   :TABLE ADD OF NEXT FREE MEMORY BLOCK
0302 50        690         LD   E,L           ;SAVE THE CONTENTS
0303 54        691         LD   D,H           :OF HL IN DE
0304 014620    692         LD   BC,MNTBED     ;ADDRESS OF LAST ENTRY IN MEMORY TABLE
0307 ED42      693         SBC  HL,BC         :HL=CURRENT LOCATION-LAST ENTRY-CARRY
0309 F21C03    694         JP   P,ALNGON      ;POSTIVE ALL MEMORY ALLOCATED SO JUMP
030C EB        695         EX   DE,HL         :TABLE ADD OF NEXT FREE MEMORY BLOCK
030D 4E        696         LD   C,(HL)        ;C=LOWER BYTE OF FREE MEMORY BLOCK ADD
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

030E 23         697           INC  HL
030F 46         698           LD   B,(HL)        :B=HIGH BYTE OF FREE MEMORY BLOCK ADD
0310 23         699           INC  HL
0311 22411F     700           LD   (MNTBPT),HL   :NEW TABLE ADDRESS STORED POINTER WORD
0314 21401F     701           LD   HL,MNTBFR     :LOCATION OF MEMORY TABLE STATUS WORD
0317 CB86       702           RES  0,(HL)        :SET STATUS WORD TO PROCESSOR # 2
0319 CB96       703           RES  2,(HL)        :NOT USING OR WAITING
031B C9         704           RET                :RETURN TO CALLING PROGRAM
031C E1         705  ALNGON   POP  HL            :HL=ADDRESS OF MEMORY TABLE STATUS WORD
031D CB86       706           RES  0,(HL)        :SET STATUS WORD TO PROCESSOR #1
031F CB96       707           RES  2,(HL)        :NOT WAITING OR USING
0321 FB         708           EI                 :ENABLE ANY DISABLED INTERRUPTS
0322 2A411F     709  ALNG01   LD   HL,(MNTBPT)   :TABLE ADD OF NEXT FREE MEMORY BLOCK
0325 ED42       710           SBC  HL,BC         :HL=CURRENT LOCATION-LAST ENTRY-CARRY
0327 FA2203     711           JP   M,ALNG01      :IF NEGATIVE REPEAT LOOP
032A F3         712           DI                 :REENABLE THE DISABLED INTERRUPTS
032B C3EE02     713           JP   ALNMEN        :JUMP TO START OF SUBROUTINE
                714  :*****************************************************************
                715  :**********SUBROUTINE DEALLOCATE MEMERY NETWORK *************
                716  :*****************************************************************
                717  :THIS SUBROUTINE DEALLOCATES A BLOCK OF MEMORY. A MEMORY
                718  :TABLE (LOMNTB) IS USED TO RETURN THE MEMORY BLOCK ADDRESS
                719  :TO THIS TABLE. THIS MEMORY TABLE CONTAINS THE START ADDRESS
                720  :OF ALL UNALLOCTED MEMORY BLOCKS. THE ADDRESS OF THE MEMORY
                721  :BLOCK TO BE PUT INTO THE MEMORY TABLE MUST BE IN THE BC
                722  :REGISTERS. THE SUBROUTINE CHECKS TO DETERMINE IF THE MEMORY
                723  :TABLE IS BEING USED. IF SO A WAIT LOOP IS ENTERED
                724  :                          LABELS
                725  :DALMEN- NAME OF SUBROUTINE
                726  :MNTBFR- PROVIDES USAGE STATUS OF MEMORY TABLE
                727  :MNTBPT- MEMORY TABLE ADDRESS OF NEXT FREE MEMORY BLOCK.
                728  :                        REGISTERS
                729  :THE SUBROUTINE DOES NOT SAVE ANY REGISTERS FROM THE CALL-
                730  :PROGRAM. THE SUBROUTINE USES REGISTERS BC,HL,DE.
                731  :                      SPECIAL INST
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

               732   :SINCE THIS SUBROUTINE CAN PUT PROCESSOR #1 INTO A WAIT
               733   :LOOP, THE CALLING ROUTINE SHOULD DISABLE INTERRUPTS
               734   :BEFORE CALLING THE SUBROUTINE
032E 21401F    735   DANMEN   LD   HL,MNTBFR     :LOCATION OF MEMORY TABLE STATUS WORD
0331 CBD6      736            SET  2,(HL)        :STATUS WORD TO PROCESSOR #2 WAITING
0333 CB4E      737            BIT  1,(HL)        :CHECK IF PROCESSOR #1 WAITING
0335 CB46      738   DANME2   BIT  0,(HL)        :CHECK IF PROCESSOR #1 USING
0337 C23503    739            JP   NZ,DANME2     :JUMP IF PROCESSOR #1 USING
033A CBC6      740   DANME1   SET  0,(HL)        :SET STATUS WORD TO PROCESSOR #1 USING
033C 28        741            DEC  HL            :HIGH ORDER BYTE OF THE MEMORY BLOCK
033D 70        742            LD   (HL),B        :ADDRESS INTO THE MEMORY TABLE
033E 2B        743            DEC  HL            :LOW ORDER BYTE OF THE MEMORY BLOCK
033F 71        744            LD   (HL),C        :ADDRESS INTO THE MEMORY TABLE
0340 22411F    745            LD   (MNTBPT),HL   :NEW TABLE ADD STORED IN POINTER WORD
0343 69        746            LD   L,C           :L=LOW ORDER BYTE OF MEMORY BLOCK ADD
0344 60        747            LD   H,B           :H=HIGH ORDER BYTE OF MEMORY BLOCK ADD
0345 21401F    748            LD   HL,MNTBFR     :LOCATION OF MEMORY TABLE STATUS WORD
0348 CB86      749            RES  0,(HL)        :SET STATUS WORD TO PROCESSOR #2
034A CB96      750            RES  2,(HL)        :NOT WAITING OR USING
034C C9        751            RET                :RETURN TO CALLING PROGRAM
               752   :*****************************************************
               753   :************SUBROUTINE LOCAL TRANSMIT QUEUE NETWORK**********
               754   :*****************************************************
               755   :THIS SUBROUTINE PUT THE MESSAGE STORED IN THE MEMORY BLOCK
               756   :ONTO THE LOCAL TRANSMIT QUEUE. A CIRCULAR QUEUE
               757   :(LOTXQ) IS USED TO STORE THE ADDRESS OF THE MESSAGE'S
               758   :MEMORY BLOCK. THE ADDRESS OF THE MESSAGE MEMORY BLOCK MUST
               759   :BE IN THE BC REGISTER. A CHECK IS ALSO MADE TO DETERMINE
               760   :IF PROCESSOR #1 IS USING THE QUEUE. IF SO, A WAIT LOOP IS
               761   :ENTERED
               762   :                          LABELS
               763   :LOTXQN-NAME OF SUBROUTINE
               764   :LTXQFR-PROVIDE USAGE STATUS OF LOCAL TRANSMIT QUEUE
               765   :LOTXEQ-ADDRESS OF LOCAL TRANSMIT END OF QUEUE
               766   :LOTXHQ-ADDRESS OF LOCAL TRANSMIT HEAD OF QUEUE
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

               767   :LOTXSQ-ADDRESS OF LOCAL TRANSMIT START OF QUEUE
               768   :LOTXTQ-ADDRESS OF LOCAL TRANSMIT TAIL OF QUEUE
               769   :                          REGISTERS
               770   :ADDRESS OF THE MESSAGE MEMORY BLOCK MUST BE IN THE BC
               771   :REGISTER PRIOR TO CALLING THIS SUBROUTINE
034D 214A20    772   LOTXQN   LD   HL,LTXQFR    :ADDRESS OF LOCAL QUEUE STATUS WORD
0350 CBD6      773            SET  2,(HL)       :STATUS WORD TO PROCESSOR #2 WAITING
0352 CB4E      774            BIT  1,(HL)       :CHECK IF PROCESSOR #1 WAITING
0354 CA5C03    775            JP   Z,LOTXN1     :JUMP IF PROCESSOR #1 NOT WAITING
0357 CB46      776   LOTXN2   BIT  0,(HL)       :CHECK IF PROCESSOR #1 USING
0359 C25703    777            JP   NZ,LOTXN2    :LOOP IF PROCESSOR #1 USING
035C CBC6      778   LOTXN1   SET  0,(HL)       :SET STATUS WORD TO PROCESSOR #1 USING
035E           779            ADDTQ LOTXSQ,LOTXEQ,LOTXTQ
035E 2A4D20    779 +          LD   HL,(LOTXTQ)  :HL=ADDRESS OF TAIL OF QUEUE
0361 71        779 +          LD   (HL),C       :PUT LOW ORDER BYTE OF THE MESSAGE
               779 +:                            MEMORY BLOCK AT TAIL OF QUEUE
0362 23        779 +          INC  HL           :PUT HIGH ORDER BYTE OF THE MESSAGE
0363 70        779 +          LD   (HL),B       :MEMORY BLOCK AT TAIL OF QUEUE
0364 23        779 +          INC  HL           :HL=NEW TAIL OF QUEUE
0365 E5        779 +          PUSH HL           :SAVE NEW TAIL OF QUEUE ADDRESS
0366 115221    779 +          LD   DE,LOTXEQ    :DE=ADDRESS OF END OF QUEUE
0369 ED52      779 +          SBC  HL,DE        :HL=CURRENT LOCATION-END QUEUE-CARRY
036B E1        779 +          POP  HL           :HL=ADD OF CURRENT TAIL OF QUEUE
036C FA7203    779 +          JP   M,A_0007     :IF SUBTRACTION NEGATIVE JUMP
036F 214F20    779 +          LD   HL,LOTXSQ    :HL=ADDRESS OF START OF QUEUE
0372 224D20    779 +A_0007    LD   (LOTXTQ),HL  :STORE VALUE OF HL INTO TAIL ADDRESS
0375 214A20    780            LD   HL,LTXQFR    :ADDRESS OF LOCAL QUEUE STATUS WORD
0378 CB86      781            RES  0,(HL)       :SET STATUS WORD TO PROCESSOR #2
037A CB96      782            RES  2,(HL)       :NOT WAITING OR USING
037C C9        783            RET               :RETURN
               784   :*************************************************************
               785   :**********SUBROUTINE TRANSMIT QUEUE NETWORK ****************
               786   :*************************************************************
               787   :THIS SUBROUTINE PUT THE MESSAGE STORED IN THE MEMORY BLOCK
               788   :ONTO THE NETWORK TO-BE-TRANSMITTED QUEUE. A CIRCULAR QUEUE
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

               789   :(NWTXQ) IS USED TO STORE THE ADDRESS OF THE MESSAGE'S
               790   :MEMORY BLOCK. THE ADDRESS OF THE MESSAGE MEMORY BLOCK MUST
               791   :BE IN THE BC REGISTER. A CHECK IS ALSO MADE TO DETERMINE
               792   :IF PROCESSOR #1 IS USING THE QUEUE. IF SO, A WAIT LOOP IS
               793   :ENTERED.
               794   :                          LABELS
               795   :NWTXQN- NAME OF SUBROUTINE
               796   :NTXQFR- PROVIDES USEAGE STATUS OF NETWORK TRANSMIT QUEUE
               797   :NWTXEQ- ADDRESS OF NETWORK TRANSMIT END OF QUEUE
               798   :NWTXHQ- ADDRESS OF NETWORK TRANSMIT HEAD OF QUEUE
               799   :NWTXSQ- ADDRESS OF NETWORK TRANSMIT START OF QUEUE
               800   :NWTXTQ- ADDRESS OF NETWORK TRANSMIT TAIL OF QUEUE
               801   :                        REGISTERS
               802   :ADDRESS OF THE MESSAGE MEMORY BLOCK MUST BE IN THE BC
               803   :REGISTER PRIOR TO CALLING THIS SUBROUTINE
037D 215321    804   NWTXQN  LD   HL,NTXQFR    :ADDRESS OF NETWORK QUEUE STATUS WORD
0380 CBD6      805           SET  2,(HL)        :STATUS WORD TO PROCESSOR #2 WAITING
0382 CB4E      806           BIT  1,(HL)        :CHECK IF PROCESSOR #1 WAITING
0384 CA8C03    807           JP   Z,NWTXN1      :JUMP IF PROCESSOR #1 NOT WAITING
0387 CB46      808   NWTXN2  BIT  0,(HL)        :CHECK IF PROCESSOR #1 USING
0389 C28703    809           JP   NZ,NWTXN2     :LOOP IF PROCESSOR #1 USING
038C CBC6      810   NWTXN1  SET  0,(HL)        :SET STATUS WORD TO PROCESSOR #1 USING
038E           811           ADDTQ NWTXSQ,NWTXEQ,NWTXTQ
038E 2A5621    811 +         LD   HL,(NWTXTQ)   :HL=ADDRESS OF TAIL OF QUEUE
0391 71        811 +         LD   (HL),C        :PUT LOW ORDER BYTE OF THE MESSAGE
               811 +:                            MEMORY BLOCK AT TAIL OF QUEUE
0392 23        811 +         INC  HL            :PUT HIGH ORDER BYTE OF THE MESSAGE
0393 70        811 +         LD   (HL),B        :MEMORY BLOCK AT TAIL OF QUEUE
0394 23        811 +         INC  HL            :HL=NEW TAIL OF QUEUE
0395 E5        811 +         PUSH HL            :SAVE NEW TAIL OF QUEUE ADDRESS
0396 115822    811 +         LD   DE,NWTXEQ     :DE=ADDRESS OF END OF QUEUE
0399 ED52      811 +         SBC  HL,DE         :HL=CURRENT LOCATION-END QUEUE-CARRY
039B E1        811 +         POP  HL            :HL=ADD OF CURRENT TAIL OF QUEUE
039C FAA203    811 +         JP   M,A_0008      :IF SUBTRACTION NEGATIVE JUMP
039F 215821    811 +         LD   HL,NWTXSQ     :HL=ADDRESS OF START OF QUEUE
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

03A2 225621     811 +A_0008  LD   (NWTXTQ),HL   :STORE VALUE OF HL INTO TAIL ADDRESS
03A5 215321     812          LD   HL,NTXQFR     :LOCATION OF QUEUE STATUS WORD
03A8 CB86       813          RES  0,(HL)        :SET STATUS WORD TO PROCESSOR #2
03AA CB96       814          RES  2,(HL)        :NOT WAITING OR USING
03AC C9         815          RET
                816          ORG  4032D
                817  :**** THIS SECTION DEFINES STORAGE SPACE FOR THE INTERRUPT ******
                818  :VECTOR TABLE.
0FC0            819  INVTAN  DEFS 8
0FC8            820  ISIO1A  DEFS 2
0FCA            821          DEFS 2
0FCC 7A02       822          DEFW RXA01
0FCE 2702       823          DEFW SXA01
0FD0            824  ICTC01  DEFS 6
0FD6 8602       825          DEFW TIME01
0FD8            826          DEFS 48
                827  :**** THIS SECTION DEFINES THE LOCATION OF THE STACK POINTER*****
1008 301F       828  SPLOCN  DEFW 8000-16
                829  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE THE     *****
                830  :STORAGE SPACE REQUIRED FOR THE NETWORK ALREADY TX QUEUE.
                831  :THE END OF QUEUE ADDRESS MUST BE THE ACTUAL END OF QUEUE
                832  :ADDRESS . TO DO THIS THE STORAGE SPACE IS
                833  :ALLOCATED BY THREE ASSEMBLIER INSTRUCTIONE. STORAGE SRACES
                834  :ARE ALSO ALLOCATED FOR THE POINTERS TO THE HEAD OF THE
                835  :QUEUE AND TAIL OF THE QUEUE.
100A            836  NATXHQ  DEFS 2
100C            837  NATXTQ  DEFS 2
100E            838  NATXSQ  DEFS 64
104E            839          DEFS 1
104F            840  NATXEQ  DEFS 1
                841  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE THE     *****
                842  :STORAGE SPACE REQUIRED FOR THE NETWORK RECEIVE QUEUE.
                843  :THE END OF QUEUE ADDRESS MUST BE THE ACTUAL END OF QUEUE
                844  :ADDRESS . TO DO THIS THE STORAGE SPACE IS
                845  :ALLOCATED BY THREE ASSEMBLIER INSTRUCTIONE. STORAGE SRACES
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND      COMMENT

               846   :ARE ALSO ALLOCATED FOR THE POINTERS TO THE HEAD OF THE
               847   :QUEUE AND TAIL OF THE QUEUE.
1050           848   NWRXHQ   DEFS 2
1052           849   NWRXTQ   DEFS 2
1054           850   NWRXSQ   DEFS 128
10D4           851            DEFS 1
10D5           852   NWRXEQ   DEFS 1
               853   :****THIS SECTION DEFINES A WORD USED FOR THE MESSAGE      ******
               854   :IDENTIFICATION BYTE. THIS BYTE IS USED TO PROVIDE A
               855   :MESSAGE IDENTIFICATION NUMBER FOR EACH NETWORK MESSAGE
               856   :TRANSMITTED
10D6 00        857   MSGID    DEFB 0
               858   :****THIS SECTION DEFINES THE PARAMETER VALUE ASSOCIATED   ******
               859   :WITH THE Z80A-SIO AND Z80A/CTC PARAMETER LIST.
10D7 FE        860   SIO01    DEFB 254D
10D8 80        861            DEFB 10000000B
10D9 18        862            DEFB 00011000B
10DA C8        863            DEFB 11001011B
10DB 20        864            DEFB 00100000B
10DC E6        865            DEFB 11100110B
10DD 00        866            DEFB 00000000B
10DE 7E        867            DEFB 01111110B
10DF 1C        868            DEFB 00011100B
10E0 00        869            DEFB 00000000B
10E1 C8        870            DEFB 11001011B
10E2 20        871            DEFB 00100000B
10E3 E2        872            DEFB 11100010B
10E4 00        873            DEFB 00000000B
10E5 7E        874            DEFB 01111110B
10E6 0000      875            DEFW 0
10E8 F8        876   CTC01A   DEFB 248D
10E9 D00F      877            DEFW ICTC01
10EB 45        878            DEFB 01000101B
10EC 02        879            DEFB 00000010B
10ED 0D        880            DEFB 00001101B
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND       COMMENT

10EE 01          881           DEFB 000000013
10EF 0000        882           DEFW 0
                 883           ORG  80000
                 884  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE THE      *****
                 885  :STORAGE SPACE REQUIRED FOR THE MEMORY TABLE.
                 886  :THE END OF TABLE ADDRESS MUST BE THE ACTUAL END OI TABLE
                 887  :ADDRESS. TO DO THIS THE TOTAL QUEUE STORAGE SPACE IS
                 888  :ALLOCATED BY THREE ASSEMBLIER INSTRUCTIONS. STORAGE SPACES
                 889  :ARE ALSO ALLOCATED FOR THE POINTER TO THE HEAD OF THE
                 890  :TABLE AND THE TABLE STATUS WORD.
1F40             891  MNTBFR   DEFS 1
1F41             892  MNTBPT   DEFS 2
1F43             893  LOMNTB   DEFS 2*128D+2
2045             894           DEFS 1
2046             895  MNTBED   DEFS 1
                 896  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE AND      *****
                 897  :DEFINE THE STORAGE SPACE REQUIRED FOR THE NETWORK ADDRESS
                 898  :TABLE THIS TABLE CONTAINS THE NETWORK ADDRESS OF ALL
                 899  :PERIPHERALS CONNECTED TO THE UNIVERSAL NETWORK INTERFACE
                 900  :DEVICE. IN ADDITION THE NUMBER OF ADDRESSES IN THE TABLE
                 901  :IS STORED AT LOCATION NWTBNU.
2047 41          902  NWADTB   DEFB 'A'
2048 31          903           DEFB '1'
2049 02          904  NWTBNU   DEFB 2
                 905  :**** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE THE      *****
                 906  :STORAGE SPACE REQUIRED FOR THE LOCAL TRANSMIT QUEUE.
                 907  :THE END OF QUEUE ADDRESS MUST BE THE ACTUAL END OF QUEUE
                 908  :ADDRESS. TO DO THIS THE TOTAL QUEUE STORAGE SPACE IS
                 909  :ALLOCATED BY THREE ASSEMBLIER INSTRUCTIONS. STORAGE SPACES
                 910  :ARE ALSO ALLOCATED FOR THE POINTERS TO THE HEAD OF THE
                 911  :QUEUE AND TAIL OF THE QUEUE AND THE QUEUE STATUS WORD.
204A             912  LTXQFR   DEFS 1
204B             913  LOTXHQ   DEFS 2
204D             914  LOTXTQ   DEFS 2
204F             915  LOTXSQ   DEFS 2*126+2
```

```
ADDR OBJECT    STMT  LABEL    OPCD OPERAND      COMMENT

2151            916           DEFS 1
2152            917  LOTXEQ   DEFS 1
                918  :***** THIS SECTION USES THE ASSEMBLIER TO ALLOCATE THE      *****
                919  :STORAGE SPACE REQUIRED FOR THE NETWORK TRANSMIT QUEUE.
                920  :THE END OF QUEUE ADDRESS MUST BE THE ACTUAL END OF QUEUE
                921  :ADDRESS. TO DO THIS THE TOTAL QUEUE STORAGE SPACE IS
                922  :ALLOCATED BY THREE ASSEMBLIER INSTRUCTIONS. STORAGE SPACES
                923  :ARE ALSO ALLOCATED FOR THE POINTERS TO THE HEAD OF THE
                924  :QUEUE AND TAIL OF THE QUEUE AND THE QUEUE STATUS WORD.
2153            925  NTXQFR   DEFS 1
2154            926  NWTXHQ   DEFS 2
2156            927  NWTXTQ   DEFS 2
2158            928  NWTXSQ   DEFS 2*1280+2
225A            929           DEFS 1
2258            930  NWTXEQ   DEFS 1
225C            931           END

TOTAL ASSEMBLER ERRORS =     0
```

SYMBOL TABLE

ADDMT1	01A2	ALNG01	0322	ALNGON	031C	ALNME1	02FD
ALNME2	02F8	ALNMEN	02EF	A_0002	015A	A_0005	0255
A_0007	0372	A_0008	03A2	B_0001	012A	B_0003	0181
B_0004	01E5	B_0006	02A9	CONMG1	01D4	CONMG2	0204
CONMG3	01E8	CONMSG	01D3	CTC01	00A8	CTC01A	10E3
DANME1	033A	DANME2	0335	DANMEN	0325	DNWRKQ	0170
DNWTX1	0111	DNWTXQ	0107	ENDCTC	00C4	ENDSIO	00A5
ICTC01	0FD0	INVTAN	0FC0	ISIO1A	0FC8	LOMNTB	1F43
LOTXEQ	2152	LOTXHQ	204B	LOTXN1	035C	LOTXN2	0357
LOTXQN	034D	LOTXSQ	204F	LOTXTQ	204D	LTXQER	204A
MNTBED	2046	MNTBER	1F40	MNTBPT	1F41	MSGIO	10D5
NATXEQ	104F	NATXHQ	100A	NATXSQ	100E	NATXTQ	100C
NMAN01	00EA	NMAN02	00F7	NMAN1A	0146	NMAN1B	0150
NMAN2A	0184	NOTMT1	01CA	NSA01	0247	NSA02	0248
NSA03	0269	NSTART	0041	NTXQER	2153	NTXSR1	02E5
NTXSR2	02ED	NWADTB	2047	NWQER	0116	NWRKEQ	1005
NWRXHQ	1050	NWRXSQ	1054	NWRXTQ	1052	NWSCA1	0195
NWTBNU	2049	NWTX01	02DF	NWTXEQ	225B	NWTXHQ	2154
NWTXN1	038C	NWTXN2	0387	NWTXQN	037D	NWTXSQ	2158
NWTXSR	02C5	NWTXTQ	2156	REPT01	0081	REPT02	0090
REPT1A	0073	RXA01	027A	RXERR1	026C	SIO01	1007
SPLOCN	1008	SXA01	0227	TIME01	0286	TIME11	028F
WDERR1	0276						

Vita

Sam Charles Sluzevich was born on 2 October 1946 in DuQuoin, Illinois. He graduated from Benton Consolidated High School in 1964. He attended the University of Southern Illinois and the University of Illinois from which he received a Bachelor of Science degree in Electrical Engineering in January 1969. Following his graduation, he entered the Air Force and received a commission in December 1969. Upon commissioning, he attended the Communication Maintenance Officer School at Keesler AFB, Mississippi. Between September 1970 and June 1973, he was assigned to Headquarters Air Force Communication Service as a communication maintenance staff officer. From there he was assigned as commander, TUSLOG detachment 150, Sahin Tepsi, Turkey. From June 1974 to May 1977 he was assigned to the Headquarters European Communication Area as chief of the wideband system division. He entered the Air Force Institute of Technology in June 1977.

Permanent address: 355 Fourth Street
Belton, Illinois 62812

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GE/EE/78-41	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PRELIMINARY DESIGN OF A UNIVERSAL NETWORK INTERFACE DEVICE		5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Sam C. Sluzevich, Captain, USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		12. REPORT DATE December 1978 13. NUMBER OF PAGES 259
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

J. P. Hipps, Major, USAF
Director of Information 1-19-79

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microprocessor
Computer communication
Computer networks
Computer interfaces

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A design was developed for a small special-purpose digital device which could be used for interfacing general peripheral devices to a communication network. The design was modularized to allow the device to be configured based upon the user's local network requirements. The design consisted of an input card, a network card and a dual processor card. A digital system life cycle

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

was used to serve as a framework for the design project. Within the life cycle, the following phases were completed: requirement definition, system design, hardware selection and design and software design. A Zilog Z80A-MCB was selected to perform the software functions and MSI circuits were used to perform the hardware functions. The software needed for the dual processor board configuration was written and assembled. The software implemented the packeting technique for message transmission.